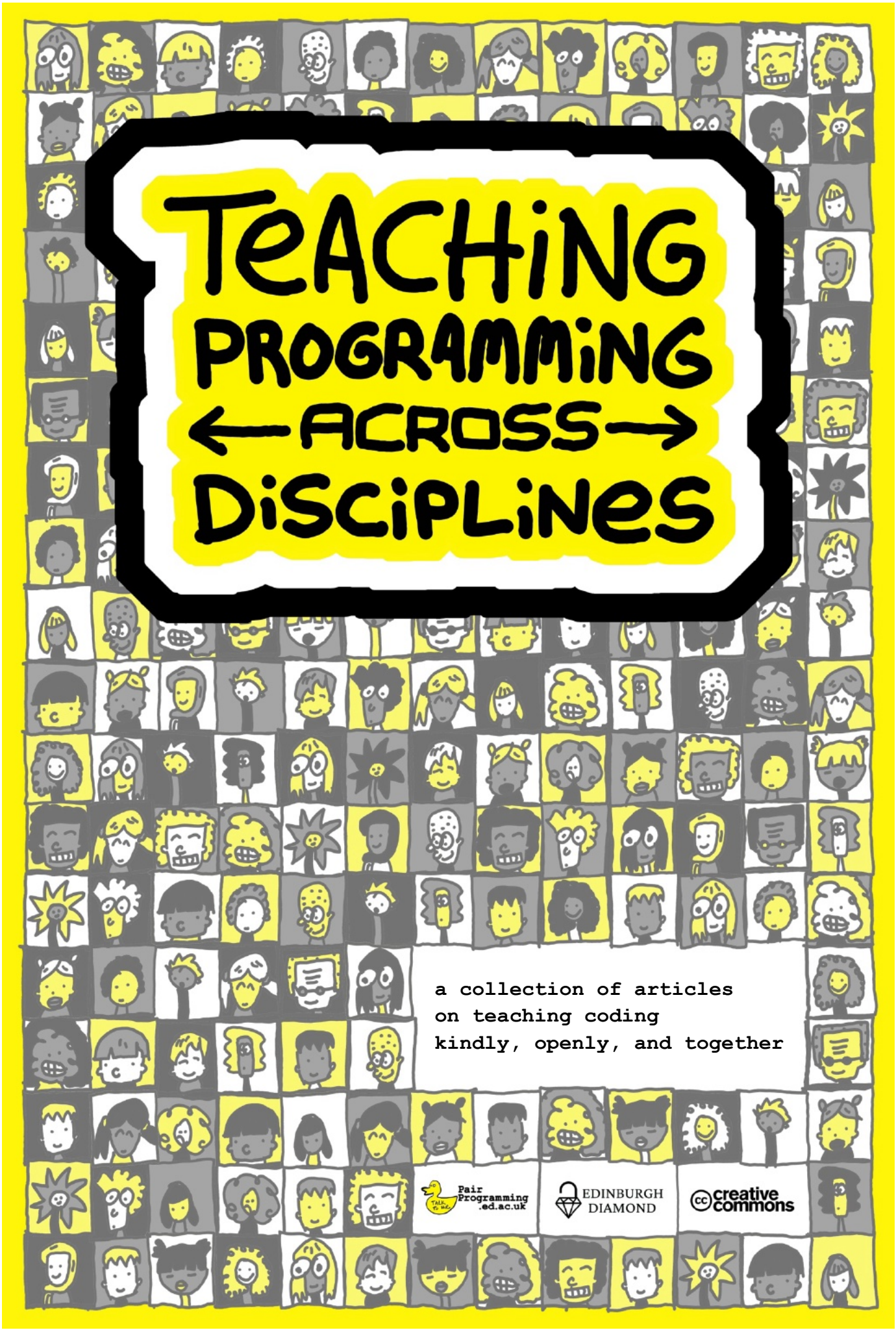
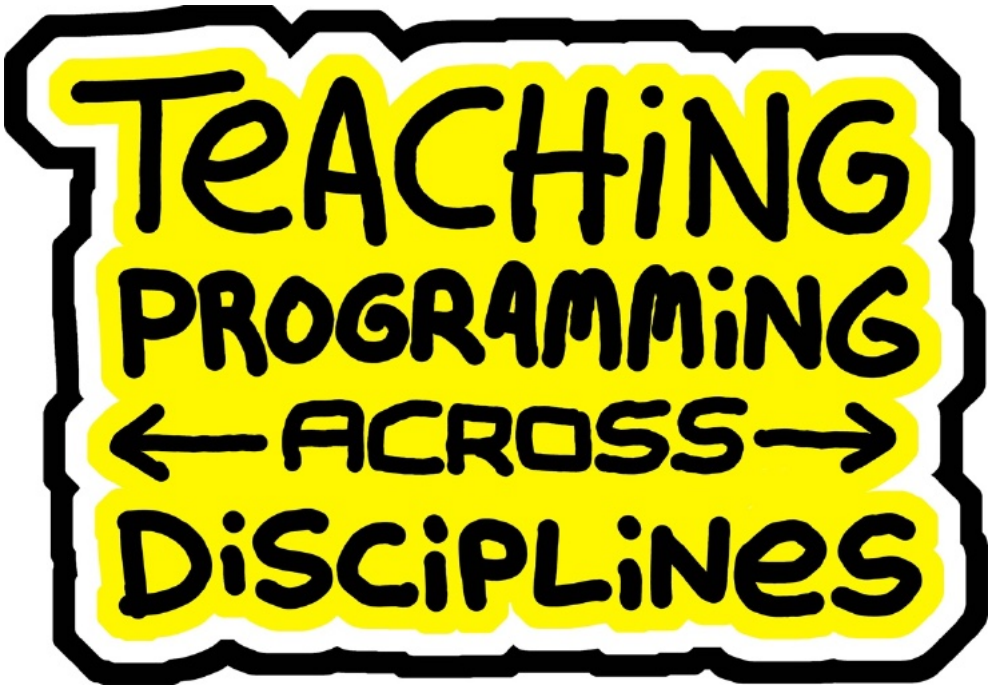

TEACHING
PROGRAMMING
←ACROSS→
DISCIPLINES
a collection of articles
on teaching coding
kindly, openly, and together
Pair Programming .ed.ac.uk
EDINBURGH DIAMOND
creative commons



Teaching Programming Across Disciplines

a collection of articles
on teaching coding
kindly, openly, and together

Co-written by 75 authors.

Edited by Brittany Blankinship, Pawel Orzechowski,
Charlotte Desvages, Kasia Banas, Umberto Noè,
Chris Oldnall, Serveh Sharifi Far,
Clare Llewellyn-MacRae, Beatrice Alex,
Ozan Evkaya and Franziska McManus.

Project coordinated by Franziska McManus.

Copyright

First published in 2026 by the University of Edinburgh

https://books.ed.ac.uk/edinburgh-diamond | @EdinDiamond[1]

Collection © Editors, 2026
Text © Contributors, 2026
Images © Contributors and copyright holders named in Image Register at the end of the book, 2026

This book is published under Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International license (CC BY-NC-SA 4.0), https://creativecommons.org/licenses/by-nc-sa/4.0/



Further details about Creative Commons licenses are available at https://creativecommons.org/about/cclicenses/

Attribution should include the following information:
Blankinship, B., Orzechowski, P., Desvages, C., Banas, K., Noè, U., Oldnall, C.A., Sharifi Far, S., Llewellyn-MacRae, C., Alex, B., Evkaya, O., and McManus, F. 2026. Teaching Programming Across Disciplines.
Edinburgh: University of Edinburgh. https://doi.org/10.2218/ED.9781836450795

Any third-party material in this book is not covered by the book's Creative Commons license. Details of the copyright ownership and permitted use of third-party material is given in the image credit lines. If you would like to reuse any third-party material not covered by the book's Creative Commons license, you will need to obtain permission directly from the copyright owner.

ISBN (print): 978-1-83645-168-6
ISBN (ebook): 978-1-83645-079-5
DOI: 10.2218/ED.9781836450795
DOI link: https://doi.org/10.2218/ED.9781836450795

Cover design by Pawel Orzechowski.

Printed and bound by BookPrintingUK.com and Lulu.com



You can read the entirety of this book online:

https://teaching-programming.github.io/book/

For updates about our community and events:

https://pairprogramming.ed.ac.uk/

This project was supported by a grant from the University of Edinburgh's Principal's Teaching Award Scheme (PTAS), which funded a project coordinator role. No authors or contributors were paid for their contribution to this book.



THE UNIVERSITY of EDINBURGH



EDINBURGH DIAMOND

[1] https://bsky.app/profile/edinburghdiamond.bsky.social

Table of contents

Copyright 1

Table of contents 3

Dedication 4

Who Are We and How Did We Get Here?.... 5

Welcome to the Book 6

Part I — Theories and Applications of Teaching Programming 8

Theme — Pair Programming: What? How? Why?

Where Do I Even Start With Pair Programming in My Classroom? A Conversation With Seasoned Practitioners 13

Structured Group Work With Assigned Asymmetrical Roles and Switching: Lessons From Pair Programming Across Disciplines 23

Pair Programming in Practice: A PhD Tutor's Reflections 27

Theme — Building Healthy Programming Habits

Brain Book Buddy Boss Break: A Practical Debugging Framework for Learners 30

A Guide on Empowering Students to Develop a Coding Mindset 35

It Depends: How to Develop Judgement in Programming 41

Theme — Programming in a World of Generative AI

Hype and The Need for Responsible Compute 48

An Optimistic Outlook on Teaching, Learning and Assessment for Coding With the Emergence of Generative AI 56

AI, Voice, and Style in Programming Education 66

How Teaching Programming Across Disciplines Can Instil Systems Thinking 68

Theme — Teaching Programming Courses: What Worked for Us

Computer Says No: Reducing Technical Barriers to Help Novice Programmers 74

Practices To Foster Inclusion and Accessibility in Programming Teaching 84

Overcoming Coding Anxiety: Lowering the Stakes and Making It Fun 93

3 Stars and 1 Wish: Small and Frequent Student Reflections Promote a Sense of Wonder and a Community of Vulnerability 99

Sequential vs. Simultaneous: Approaches to Learning Programming and Statistics 109

Part II — Teaching Programming in Context: Design and Practice..... 114

Theme — Drawing on Group Dynamics for Programming Education

Peer Programming in Action: Pair Programming in Larger Groups 118

Leveraging the Heterogeneity: Teaching Computing Skills to a Multidisciplinary Cohort With a Variable Skill Level 125

Learning Together Across Modes: Online and On-Site Pair Programming in a Fusion Course 135

Theme — Teaching Programming to Novices: Python and Beyond

A Practical Guide to Teaching Python as a Computational Tool in an Introductory Data Analysis Course 143

Notebooks for Novices? Pros and Cons of Jupyter 147

Bridging Languages: Teaching C to Python Novices 153

Theme — Thinking Outside the Box: Designing Engaging Learning Experiences

Dungeon Crawlers and Anarchists: Co-Designing Programming Teaching Experiences 160

Removing Barriers by Programming Without Computers 166

Practical Approaches to Using Sound and Music in Programming Pedagogy 171

Ways to Teach Online: Lessons Learned Through Experience 181

Seeing Before Coding, Doodling Before Doing: How Teaching Data Visualisation Transforms the Way Students Think 189

Part III — Assessment and Feedback: Principles and Practice 199

Theme — Frameworks for Assessment and Feedback

Developing a Skills-Based Framework for Assessments in Programming Courses 202

Managing the Rubik's Cube of Assessment: The Action Feedback Protocol 208

Theme — Practical Methods for Assessing Programming

Assessment and Marking in an Introductory Programming Course for Physics Students 213

Generative AI in Assessments: Before, During, and After 218

Using Automated Marking in Programming Courses To Enhance Learning 225

Part IV — Case Studies 231

Theme — Building Learning Communities and Resources

A Gentle Introduction to Coding: Narratives From a Programming Summer School for Social Sciences and Humanities 233

Case Study From Scotland's Rural College: The Wild(Life) Side of Pair Programming 240

Interdisciplinary by Design: The Centre for Data, Culture, & Society Training Programme 243

Changing Civil Engineering Students' Mindset Toward Programming 248

Lost in Translation: Complexities and Good Practices of Translating Coding-Based Teaching Materials Across Multiple Human Languages 254

Closing Remarks for This Book 262

Authors' Biographies 263

Image Register 289

Dedication

This book is dedicated to Ada Lucy Banas, who entered the world on St Patrick's Day, 17 March 2026 ☘️, in the middle of the book development process.

Who Are We and How Did We Get Here?

Brittany Blankinship

Edinburgh Medical School, University of Edinburgh
b.blankinship@ed.ac.uk

Franziska McManus

Edinburgh Medical School, University of Edinburgh
z.mcmanus@ed.ac.uk

Pawel Orzechowski

Edinburgh Medical School, University of Edinburgh
pawel.orzechowski@ed.ac.uk

The origin of this book can be traced to one afternoon in a University of Edinburgh cafe that no longer exists. Sat around a table were colleagues who teach programming to students in medicine, business, mathematics, and psychology. Amongst the buzz of university staff and students working, having meetings, and enjoying a hot beverage, we discussed ideas for our fledgling community. It was on this auspicious afternoon that Kasia Banas suggested "What if we organize a small workshop or conference? Like a summer school but in winter?" And so, the Edinburgh Winter School was not quite born, but the seeds planted for what would grow into the annual Winter School and eventually this very book.

Some months prior, we had started a blossoming community of support and practice anchored around the use of using pair programming in teaching and learning (hence our community's email account and website – pairprogramming@ed.ac.uk[2] and https://www.pairprogramming.ed.ac.uk respectively). We had identified a gap in teaching-focused communities of practice: educators such as ourselves who love teaching and find joy in it but teach programming across disciplines. And in this, we were all facing similar problems: how to teach practical skills in large courses to diverse groups of students, many of whom did not begin their studies to learn or develop programming skills.

This book and our community have expanded beyond pair programming, but it remains core to who we are. As such, our book starts with pair programming, sharing our various experiences using it as a structured form of group work in programming courses. Through pair programming, we aim to give students a learning experience where coding is a joyful, creative, and community-centered activity (instead of scary, frustrating, and lonely). This continues to be a golden thread connecting our community – hence the title "Teaching Programming Across Disciplines: A collection of articles on teaching coding kindly, openly, and together".

[2] mailto:pairprogramming@ed.ac.uk

Welcome to the Book

Franziska McManus

Edinburgh Medical School,
University of Edinburgh
z.mcmanus@ed.ac.uk

Brittany Blankinship

Edinburgh Medical School,
University of Edinburgh
b.blankinship@ed.ac.uk

Pawel Orzechowski

Edinburgh Medical School,
University of Edinburgh
pawel.orzechowski@ed.ac.uk

Welcome to the Teaching Programming Across Disciplines Book! Our book is an edited, open-access volume comprised of many short chapters written by groups of authors from our community. In this first edition of the book, we have curated 36 chapters from 75 authors. Since the original pitch, these chapters have been drafted, revised, and polished, with rewrites from authors, feedback from the editing team, and many writing retreats along the way. Contributions to this book are not limited to one institution, and while reading you will notice that our authors have drawn on their experiences across different institutions, countries, and educational backgrounds.

Come on a journey through some of the core questions that arise when teaching programming across disciplines. The book is grouped into four sections (Part I, Part II, Part III, and Part IV), each comprised of themes, which themselves contain chapters. Chapters are the smallest unit in the book and vary in length. You will find teaching resources spread across every chapter. Some chapters you can even share with your students directly. Unless otherwise specified, the illustrations can be used as an additional teaching resource and to share your practice [3].

Why Did We Write This Book?

At a basic level, we wrote this book because it is something we would have wanted to exist when we started our own teaching journeys. That is, we would have liked guidance from a community with specific experience in teaching programming to non-programming (i.e., not computer science) students.

There is an existing literature on teaching programming within computer science. However, the teaching advice and challenges are not always relevant to our students or contexts. For example, a fundamental challenge of teaching programming across disciplines is its status as a supplementary tool alongside a primary discipline. Within a broader programme of study, our courses are often given limited time and are taught in conjunction with other skills (e.g. statistics, research design, laboratory work). This creates unique challenges related to teaching to a tight timeframe, teaching many skills at once, and managing student expectations and confidence. We are not teaching coding as a skill in and of itself. It has to exist rooted within the disciplinary context and alongside it.

Our community works in departments related to our disciplines, rather than the programming we teach. This often results in silos – or even teams of one – delivering the entire programming curriculum for a cohort. As a result, we rely on chance meetings with colleagues at conferences and coding clubs to create opportunities for likeminded people to connect and develop their practice. We hope that this book will offer something similar - a window into how others teach and an opportunity to pick the best bits for your own teaching. Above all, we hope that it will be a source of support and solidarity that speaks to your unique teaching experiences.

Who Is This Book For?

This book is for anyone who is interested in teaching, programming, or the intersection of these ideas. Many chapters are written with the aim of creating community and sharing our experiences with fellow educators. Others are suitable for students to read as they are.

The volume we have edited is relevant to both educators starting out on their teaching journey and more experienced

[3] Just follow licensing outlined in the Images Section and credit the creator

teachers. We like to think that there is value for teaching-adjacent colleagues as well – leadership, administrators, etc.

How To Read This Book

There are many (we have yet to quantify an exhaustive list) ways in which a reader can navigate our book. It is unlikely that you will open this book and read it cover to cover. To find chapters most relevant to your teaching, you can use the Part and Theme names as a guide. Alternatively, close your eyes and randomly open the physical book to a page or scroll on your computer and see where your cursor lands on our online version of the book.

A Note on Vocabulary

We would not be a community of academics working on a project together if we did not do the very academic thing of defining our terms. At some higher education institutions, the terms “course” and “module” are used to refer to the overall degree program of study and individual organized learning opportunities bearing credit, while others use “program” and “course”. Similarly, you may notice “student” and “learner” are both used. As editors, we have left it to the authors’ discretion as this is most representative of how the terms are used across our community.

We hope you find something interesting to you as a human, and perhaps even useful if you are an educator, in the chapters that follow. Happy reading!

Part I — Theories and Applications of Teaching Programming

Brittany Blankinship

Edinburgh Medical School, University of Edinburgh
b.blankinship@ed.ac.uk

Franziska McManus

Edinburgh Medical School, University of Edinburgh
z.mcmanus@ed.ac.uk

Umberto Noè

School of Philosophy, Psychology and Language Sciences, University of Edinburgh
Umberto.Noe@ed.ac.uk

As a community, our philosophy towards teaching is grounded in the practice and framework of pair programming (if you haven't heard of this yet, read on!). This framework for structured group work, taken from industry, was the first aspect of our teaching which we shared amongst ourselves at the University of Edinburgh and became a uniting feature of our teaching. Today, pair programming features in all of our teaching as a standard. Our teaching and working life centres around the philosophy of working together to create a shared product and vision – that is, working openly, kindly, and together.

In the first theme, you will get acquainted with what pair programming is and how it can be implemented across courses, levels of study, and disciplines. By the end of it, we hope it will convince you that pair programming is a framework worth getting excited about and perhaps something to implement in your own teaching.

The second theme focuses on a selection of core concepts in programming that are essential for building healthy programming habits. While learning a programming language's vocabulary and syntax is a fundamental part of learning to code, these chapters propose broader approaches that nurture more general skills required to be successful coders and learners (hello transferrable skills).

The third theme acknowledges that being a good learner necessarily means thinking critically and making connections across ideas and topics. As educators, we expect our students not only to produce code that executes, but also to interact meaningfully with it and the wider context. It discusses what it means to teach and learn programming in a world of Generative AI, addressing the good, the bad, and the ugly. You will not find another debate but a range of viewpoints as they exist in our teaching community. These chapters will resonate with educators, address concepts that have always been fundamental to programming, and hopefully reveal a few surprises along the way.

Finally, as programming educators, you may have an idea of what concepts and mindsets you would like to teach your students. But how do you actually implement them in a pragmatic, engaging, and effective way? In the fourth and final theme of Part I, our authors tackle core pedagogical issues from the perspective of teaching programming:

- How do we get total beginners learning a tool they never expected to use in the first place?
- How do you teach multiple complex concepts at the same time?
- How do you build confidence and create an inclusive and welcoming atmosphere for your students?

Some of the concepts and ideas presented in Part I may be familiar to you already, but we hope that you will gain either a framework to describe what you are already doing or a new way to approach teaching.

Theme — Pair Programming: What? How? Why?

Where Do I Even Start With Pair Programming? (Orzechowski, Blankinship, and Banas 2026)

An introduction to pair programming in the form of a (written) podcast interview where you can read or listen. This chapter is a joyful and pragmatic approach to answering some of the most frequently asked questions around introducing pair programming in the classroom.

This chapter is for you if you are interested in pair programming as a teaching practice or you would like ideas for how to explain it to your students or colleagues.

Lessons From Pair Programming Across Disciplines (Desvages et al. 2026)

A walkthrough of the origins and practice of pair programming and what this has looked like for the authors in their disciplines. A highlight of this chapter is the fun illustrations that bring to life explanations of pair programming supported by student feedback.

This chapter is for you if you want to learn more about pair programming or are looking for a resource to explain pair programming to students, colleagues, and management.

A PhD Tutor's Reflections (Elliot 2026)

A PhD students' experience of teaching pair programming in an introductory R course. The author reflects on when pair programming works and when it doesn't, as well as the tutor's role in facilitating pair programming.

This chapter is for you if you are a graduate student or tutor on a course that uses pair programming, or are in a position to implement it in a tutorial setting. Course organisers will also benefit from the insightful reflections from the front line of teaching.

Theme — Building Healthy Programming Habits

A Practical Debugging Framework for Learners (Doig 2026)

An accessible and joyful introduction to debugging via the 5 B's framework. The sections of Brain, Book, Buddy, Boss, Break are expertly scaffolded and explained, turning this chapter into an adaptable teaching resource.

This chapter is for you if you teach any programming language and want your students to approach errors and debugging in a critical and independent fashion.

Empowering Students to Develop a Coding Mindset (Bray et al. 2026)

The most important skillset you can learn when coding is not the syntax of the specific programming language, but rather the "coding mindset". This chapter outlines practical suggestions and approaches for ensuring your teaching goes beyond syntax and focuses on developing students' mindset to allow them to effectively code in other languages in the future.

This chapter is for you if you are embarking on teaching your first programming course and want to develop the course beyond the syntax you might be teaching. Experienced programmers will also find the chapter interesting as a source for reflection on their own "coding mindset" and how it has developed.

It Depends: How to Develop Judgement in Programming (Wilson and Blankinship 2026)

Extending the idea of "best practice" in coding, this chapter asks the programmer to question the needs of their context rather than seeing "good" and "bad" coding practices in black and white. The central concept, that there are "forces" of coding, will be useful as

a teaching resource to help students write good code and consider why they are coding in the first place.

This chapter is for anyone who codes and wants to expand their approach to coding independently, consciously, and/or collaboratively.

Theme — Programming in a World of Generative AI

Hype and The Need for Responsible Compute (Ahern, Skipsey, and Riviera 2026)

Every new computational tool or AI model comes with exciting promises, but it also brings challenges. These often leave behind questions about fairness, who benefits from this technology, and who pays the hidden costs. As educators, we are placed in the middle of this hype, trying to help our students make sense of these tools.

This chapter will take you through the history of responsible computing through the lens of the United Nations (UN) Sustainable Development Goals. You will learn how issues like inequality, environmental harm, workforce exploitation, and bias can be part of new technologies. Along this journey, the authors share classroom activities, teaching ideas, and recommended readings you can assign in your own courses. This chapter is for you if you want your students not only to learn about fancy new algorithms, but also to be responsible and sustainable in their usage.

Teaching With the Emergence of Generative AI (Evkaya et al. 2026)

You are likely familiar with the fact that students can now generate working code in a matter of seconds with AI. However, you may also worry about what this means for assessments and question the value of your own teaching. This chapter invites you to rethink your approach to teaching and focus on what matters most when teaching students how to code. The authors talk about how Gen AI tools can help with exploration and feedback, while keeping some space for struggle and genuine understanding. They share examples of how to adapt your teaching and assessment without giving up on rigour.

If you are wondering about how to rethink your teaching or assessment approach for students coding with AI, this will be an interesting read.

AI, Voice, and Style in Programming Education (Wilson 2026)

The author puts into writing many facets of the impact of Generative AI on academic skills development that educators will have been uncomfortable about but may not have been able to put into words themselves.

This chapter is for educators who want to be able to communicate their concerns around the erosion of academic skills with the existence and accessibility of Generative AI. It will be a useful teaching resource to give to students as is.

How Teaching Programming Across Disciplines Can Instil Systems Thinking (Riviera et al. 2026)

Starting with the purpose and function of the university as an institution, this chapter takes you on a journey through the benefits of learning to program from a systems thinking perspective. If systems thinking is a new concept for you, you will find an intriguing introduction and a new way of looking at programming and the wider world (i.e., systems).

For many readers, this will answer the question of why it is a worthwhile endeavour to teach students programming skills (that they may not use again). As a teaching resource, this chapter would be excellent compulsory reading for students and a starting point for discussion-based tutorials.

Theme — Teaching Programming Courses: What Worked for Us

Reducing Technological Barriers in Programming Courses (Wadsworth et al. 2026)

Learning to code for the first time is fraught with challenges ranging from the technical, cultural, and practical. As an educator, how do you help your students when “the computer says no”? This chapter covers implementable steps for reducing barriers in programming courses, to ensure students don’t lose motivation to learn programming when faced with such issues, while ensuring instructors are not overworked.

This chapter is for you if you teach novice programmers and want to reduce friction in getting set up and working through the programming curriculum. With clear suggestions and a summary of pros and cons, it will allow you to pick the best setup for your own course!

Inclusion and Accessibility in Programming Teaching (Colquhoun et al. 2026)

Moving beyond standard prescriptions for inclusion and accessibility, this chapter takes a broader approach to pedagogical practice that fosters learning, allows vulnerability, and creates a space for all students to succeed. It provides a structured overview of the impact of different methods of delivery and a range of approaches that you can use dynamically depending on your student cohorts.

For programming educators who would like to make their classroom more welcoming, this chapter will offer practical approaches tailored to the experience of teaching programming across disciplines and being an adaptive teacher.

Overcoming Coding Anxiety (Oldnall et al. 2026)

For many students, anxiety is a central and paralysing part of their coding experience. This is particularly true for students in non-programming disciplines where educators may need to make a targeted effort to approach this head on. The authors offer practical suggestions and tips provided from their experiences that you can easily pick up and embed out of the box in your own teaching.

This chapter is for you if you have ever taught coding to anxious students or would like ideas on ways to make your programming course more accessible and fun for all students, regardless of background.

3 Stars and 1 Wish (Orzechowski et al. 2026)

With its origins in primary school teaching, this reflective practice framework will be a surprising and adaptable tool for use in higher education teaching, both in-person and online. This chapter introduces the idea of ‘3 stars and a wish’, where reflection is a flexible way of creating community that may replace existing discussion boards and feedback surveys. The authors present a case study and practical tips for implementing the framework.

This chapter is for you if you are an educator who would like to integrate continuous bite-sized reflection into your teaching or for anyone who is looking for ways to pulse-check student learning and progress.

Sequential vs. Simultaneous (Young et al. 2026)

Within many statistics modules, programming is presented as a tool which students need to learn quickly, engage with briefly, and then move on from. Yet, both statistics and programming come with a heavy cognitive load and are sources of angst, anxiety, and confusion for students. This chapter compares three approaches to teaching statistics and programming for students who need to know both. This chapter is for anyone designing (or re-designing) a course teaching both programming and statistics. With concrete examples from the author’s diverse experiences, this chapter presents a practical guide as well as an opportunity to learn from others who have been there before.

References

Ahern, Samantha, Sam Skipsey, and Leo Riviera. 2026. “Hype and the Need for Responsible Compute.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Bray, Neil, Samantha Alvarez Madrazo, Lucia Michielin, Nick Jayanth, and Sam Skipsey. 2026. “A Guide on Empowering Students to Develop a Coding Mindset.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Colquhoun, Rebecca, Samantha Ahern, Brittany Blankinship, and Patricia Loto. 2026. “Practices to Foster Inclusion and Accessibility in Programming Teaching.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Desvages, Charlotte, Brittany Blankinship, Umberto Noè, and Pawel Orzechowski. 2026. “Structured Group Work with Assigned Asymmetrical Roles and Switching: Lessons from Pair Programming Across Disciplines.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Doig, Mar. 2026. “Brain Book Buddy Boss Break: A Practical Debugging Framework for Learners.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Elliot, Sarah. 2026. “Pair Programming in Practice: A PhD Tutor’s Reflections.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Evkaya, Ozan, Hebatallah Shoukry, TJ Elmas, Laila Dabab Nahas, and Steven Watterson. 2026. “An Optimistic Outlook on Teaching, Learning and Assessment for Coding with the Emergence of Generative AI.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Oldnall, Chris, William Kay, Chris Sutherland, Tiago A. Marques, and Robert Young. 2026. “Overcoming Coding Anxiety: Lowering the Stakes and Making It Fun.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Orzechowski, Pawel, Brittany Blankinship, and Kasia Banas. 2026. “Where Do i Even Start with Pair Programming in My Classroom? A Conversation with Seasoned Practitioners.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Orzechowski, Pawel, Elaine Mowat, Karim Rivera Lares, Rebecca Sewell, Clare Llewellyn-MacRae, Bea Alex, and Kasia Banas. 2026. “3 Stars and 1 Wish: Small and Frequent Student Reflections Promote a Sense of Wonder and a Community of Vulnerability.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Riviera, Leo, Maeve Murphy Quinlan, Rebecca Colquhoun, and Devanjan Bhattacharya. 2026. “How Teaching Programming Across Disciplines Can Instil Systems Thinking.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Wadsworth, Lizzie, Lucia Michielin, Chris Cooling, Ozan Evkaya, Sam Skipsey, and David Cutting. 2026. “Computer Says No: Reducing Technical Barriers to Help Novice Programmers.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Wilson, John. 2026. “AI, Voice, and Style in Programming Education.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Wilson, John, and Brittany Blankinship. 2026. “It Depends: How to Develop Judgement in Programming.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Young, Robert S., Rebecca L. Colquhoun, Tiago A. Marques, Brittany Blankinship, Ozan Evkaya, and William P. Kay. 2026. “Sequential Vs. Simultaneous: Approaches to Learning Programming and Statistics.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Where Do I Even Start With Pair Programming in My Classroom? A Conversation With Seasoned Practitioners

Pawel Orzechowski

Edinburgh Medical School, University of Edinburgh
pawel.orzechowski@ed.ac.uk

Brittany Blankinship

Edinburgh Medical School, University of Edinburgh
b.blankinship@ed.ac.uk

Kasia Banas

Edinburgh Medical School, University of Edinburgh

Context

In this chapter we present a conversation between three educators who have been using pair programming for many years. They will introduce the practice and answer frequently asked questions about using pair programming in education.

You can also listen to the audio podcast version of this interview[4]. Thank you to Miłosz Karpowicz and Jan Banaś for audio editing advice and help.



Figure 2.1: Authors recording the podcast version. We rigged the space for optimal audio quality and comfort.

Communicating the Why

Let's Jump Right In:

Interviewer (Kasia): Hi, we are a group of educators at the University of Edinburgh. We have been using an asymmetrical group working technique in our classrooms, called Pair programming. It's used a lot in the industry, but not so much in education. Recently, a lot of colleagues have become interested in our experience of this, asking us questions at conferences and within our own university.

I've decided to interview my team who are experienced with pair programming in teaching. They will answer many of those common questions, and we hope this will benefit the educators who want to know how to start using Pair Programming in their classrooms.

Pawel: Hi, my name is Pawel and I have been teaching programming for almost a decade. I've always used Pair Programming in my teaching, and at times in my programming job before that. At the university I taught in Business School and Medical School, but also helped colleagues in many other departments. I love working with students on the beginning of their coding journey. They bring a lot of their own stories and thinking patterns to the table.

Brittany: Hello, my name is Brittany and I have been teaching programming and programming related topics for around 7 years in online and hybrid settings. More recently my teaching has focused on online masters courses and introductory courses. In my opinion, learning is a joyful experience and I am passionate about creating fun and joyful learning opportunities for my students, which pair programming definitely contributes to!

Kasia: Hello, my name is Kasia and I have been teaching programming for 5 years now. I also use pair programming in my courses, particularly those I teach in person. Join me over the next while as I interview my colleagues and ask them some of the questions that frequently come up when we introduce pair programming to other people.

What Is Pair Programming?

Interviewer: Let's start with the basics: What is Pair programming (or pair-groupwork in general) and how does it differ from just working in a pair?

[4] media.ed.ac.uk/media/1_xnjvow1u



Figure 2.2: In pair programming two students work on one computer: Driver operates the computer, while Navigator helps them to find problems and offers advice.

Pawel: It's a new way to work on practical tasks, where students work in pairs and take turns. One of them is performing the active, leading role of a Driver, and the other one is supporting and guiding them as a Navigator. They switch after every task, or every 15 minutes, so both can experience driving and navigating. That's because there's a lot to learn from leading and from supporting. Communication is the key here, explaining what you're doing is a great way to learn.

Brittany: The role names come from the idea of driving a car - only one person is behind the wheel actively moving the vehicle, but the navigator in the passenger seat is crucial to making sure they reach the intended destination ("turn right here", "take that road"). In practice when pair programming, you would work on one computer, and pass the keyboard around, so that only the driver can 'DO' things like click or type. This means that when pair programming, you have to fully embrace your current role.

When it's time to switch roles you would just pass the keyboard around. Or a whiteboard marker, or whatever you are using to complete the task. So the core elements are: we take roles, we switch. But it's not like the navigator takes a break - both roles are active and have different responsibilities, they are asymmetrical.

Pawel: And what is really interesting is that this idea did not start in academia as a way of teach or learn. It comes from Agile Software Development practices, and programmers have been using it for a while when they work. Coding involves a lot of problem solving and that's easier to do when you can discuss it with someone. But when no-one is around, you talk to yourself, or even to an inanimate object like a rubber duck.

Interviewer: Oh, yeah, I've seen a lot of rubber ducks on the desks of programmers. There's that joke that it's good to explain your code to someone, but if there is no one around you can talk to the duck?

Brittany: Yes, they come from the same idea. I often tell my students that it is useful to talk out loud and sort through your ideas - talking is a way of thinking. In programming we have this idea of rubber-ducking - talking to an inanimate object like a rubberduck (I have a wee ghost on my desk not an actual rubber duck), my colleagues have them as laptop stickers, or keyrings. With PP your rubber duck is a human who can talk back to you! Making it even more impactful, because most rubber ducks don't talk back! haha.

Interviewer: Ah, this sounds really interesting. And we said before that programmers in the software industry use this a lot. Has it been used in teaching? Have people found that pair programming can be beneficial in the classroom?

Pawel: It's quite a young technique in education, it's been gaining traction over the last decade or so. There is some recent research that using it in the classroom produces better code and more discussion between students, when comparing it to solo programming. Another team (Goel and Kathuria 2010) found that it is particularly beneficial for students with lower prior programming skills.

Brittany: We've been using it across our departments for about a decade, with about two thousands students in total. We've gathered a lot of expertise and are reaching a place where we're starting to look more analytically into what works and what doesn't. Especially what types of pair programming give people a better learning experience and better outcomes.

Why PP Is Good for Learning

Interviewer: That's right, it is all about learning better after all. So how does pair programming fit into your courses? Where does it shine and where does it not?

Brittany: One thing's for sure: practical tasks is where working in pairs shines the most. When our students engage with a new creative task (like programming, problem solving or design) there's so much to think about. You're trying to work with the tools, but also keep in mind the context, and also incorporate whatever you learned recently. There is trial, and error, and exploration, and correcting your own mistakes. That's where working in pairs, as with a driver and navigator, works best.

Pawel: Imagine you're going on a quest, but not alone. There's always your trustworthy sidekick, your supporting character. Always on your flank, there to catch you if you stumble. It is easier to stay brave, and keep up the stamina when you know someone will swap you in a few minutes. Tasks where support matters, they work really well.

But yeah, some things do not work too well in a pair - things like reading paragraphs of text or watching videos. Any time when you're just passively consuming content. Maybe that's because everyone reads at a different speed, and takes notes differently (Orzechowski et al. 2026). We build our whole courses around this: students 'consume' content individually at home before the class. And then we meet all together for the practical exercises - those happen in pairs, in the classroom.

Interviewer: That's called a Flipped Classroom, right? Where instead of listening to the teacher together, and then practicing alone, the format is flipped. So students watch pre-recorded lecture videos alone, and then come to practice together with the teacher.

Pawel: Totally, it gets even better - you come to practice not just with the teacher but also with all your fellow students. Everyone becomes a little bit of a teacher, and a little bit of a student. It's like we do not have a classroom of

40 students and 1 teacher... It's more of a classroom with 41 teachers - everybody contributes! And that's where doing practical things works well - no one in the class has a full grasp of the concept yet, but we will all figure it out together.

Interviewer: That was meant to be my next question: some students will grasp the key concepts sooner than others, or some may have had previous exposure to the course content. Would doing pair programming with a group of mixed-skill students be more difficult? What if the two students in a pair have vastly different levels of experience?

Pawel: That's the really surprising thing, actually! It is just the opposite! Our students are here to learn, and the best way to learn something is to try to explain it to another human. When two people engage on that journey, they have to synchronise on what they believe is correct.

And yes, actual correctness is important too, you need quick ways to check if your solution works. That's why tutors are constantly visiting each group, throughout the class, and we also write our code so that it asserts its own correctness. But the bulk of the work happens with two students immersing themselves in a challenge as a team, trying to learn the most, to teach each other the most.

Interviewer: Ok, but doesn't this result in students working slower than they would on their own? You don't think that's a problem?

Brittany: We optimise for learning practical skills and being able to do it by yourself, a month later, with minimal support from notes or colleagues. But to get there, first everyone needs to build their muscle memory. And everyone is the operative word here - it's not just about YOU learning. You are a part of a cohort, and we are optimising for everyone in the group getting as much from the course as possible.

This type of community spirit, and empathy towards your peers, is really compatible with working in pairs. Members of the pair take responsibility for each other. But yes, there is some expectation management needed - you're not there to solve the puzzles as quickly as possible, you're there to learn and to help others learn and benefit from the opportunity of having another person to work with, rather than learning alone.

Interviewer: Oh, right, this reminds me of this approach in medicine, where they tell the students to "see one, do one, teach one". Young doctors would first watch a procedure being done; then they do it themselves under supervision, and then train others to do it. Only when you can guide someone else, you really understand it, right?

Pawel: Yeah, absolutely! When done well, pair programming puts you in all of those roles many times within each hour: you see, you do, and you teach. You get to see the problem from many angles, and also switch between learning in a passive, active, and reflective way. There's nothing new in this idea, but what's new is how quickly we get to switch and learn. You're correct - overall the process, and the learning, is more valuable than the output.

Interviewer: Speaking of value, I totally understand why in academia we optimise for learning. But you mentioned that this practice comes from the industry. How do they justify using two people to do a job that one person could do, and potentially even faster?

Pawel: It appears to be puzzling at first, right? Counterproductive, using two people when one would do! But this false paradox comes from badly measuring what is good, what is efficient. The best writer or programmer is not the person who writes most book pages or most lines of code. Rather, the best writing fulfills its intended purpose and does it robustly. Extra points if it can be reused later for another purpose.

In creative industries, like coding, we can't think about efficiency like we do in manual labour, it's not like laying bricks when you build a house. A better metaphor would be that coding is like optimising a restaurant, so that food is tastier, and clients happier. It's about working smart, not working hard. Coding is not a repetitive, replicable task like peeling potatoes or assembling cars. Coding is about quality, efficiency, and internal communication. So two people can be justified if what they create is more future-proof and better integrated.

Brittany: And because each coding challenge is slightly different, and parts are so interconnected... it makes sense for two people to work together, and constantly check each other's work. The more something is interconnected, the more dramatic the impact of errors can be. We need code to be of super super high quality and that's worth every minute every person spent on it.

It depends on a task, but often another pair of eyes makes the code more solid, which is worth it in the long run. Additionally, pairing up and working with people across the team means more people have a deep understanding of what we're building, which makes for a more robust and agile team in the long run.

Students' Reactions

Interviewer: Oh wow, I see how this approach would be useful, both in education and in industry. But do students get it? Do you have to spend a lot of time convincing them?

Pawel: Yeah they get it. They say that in a successful company 30% of the budget should be spent on marketing. We sort of do the same with our teaching methods. We spend a lot of time and energy explaining to students why pair programming is a great way to learn. Pretty much like we did to you, just now. Students are here to learn, and they pick up good learning practices very quickly.

Depending on the course, sometimes the benefits take a session or two to become obvious. But it always happens in the end. The persistence of the teaching team and reiterating the 'why' is needed at the beginning, but once the wheel is spinning, once everyone sees the benefit, it just keeps giving. What you get quite soon is the excited buzz in the classroom as pairs code together. It's invigorating.

Interviewer: Have you ever encountered students who didn't want

to be put in pairs, and insisted on working alone? What do you do then?

Brittany: Weirdly, this happens very rarely. Occasionally we put people in 'pairs' of 3... it is especially useful when someone didn't complete the pre-reading, or is anxious, or has technical troubles. And learning something new is quite often anxiety inducing, but having a familiar structure and clear expectations can be actually useful for calming the nerves.

Being at university, or even learning in general, is a social experience, so the fact that people are all different is great! This is one of the benefits of pair programming - interacting with people from different cultures, languages, backgrounds. Differences between you and your partner are a benefit, not a problem. The programmers would say "it's a feature, not a bug (haha)".

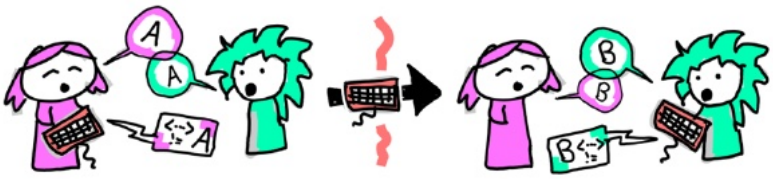


Figure 2.3: Students swap roles frequently to have experience of driving and navigating

Preparing for the Session

Interviewer: Ok, this sounds really cool. Can we now talk a little bit about the practicalities? Walk me through how pair programming actually happens in the classroom.

How To Run a PP Session

Interviewer: For example, how long would your pair programming sessions be?

Brittany: In our teaching, each session is between 1 and 3 hours long. It should not be shorter than 1 hour for just coding, since students need time to 'settle into their chair', open the laptop, open software, say hi to their partner, etc. But it could be longer - it really depends on the number and types of tasks. It's also quite important to suggest to students that they should take breaks, particularly if the session is 2-3 hours long - we usually just tell pairs to manage their own time and take a break somewhere in the middle.

Interviewer: And what is the ideal class size for pair programming? How many pairs and how many teachers would you usually have?

Pawel: When we teach in-person, we are often limited by the room capacity, but online the only limitation is the size of a teaching team. My Business school course had 160 students. That's a lot, but I've split them into four groups things became very manageable. Each session had 40 students, me and 1 teaching assistant. It was a very good size. It was a bit taxing on the teaching team to run around for 2 hours, four times a week. But the benefits were enormous, and there were never too many hands in the air or a long queue of people waiting for help.

I would say 1 staff member can easily provide help to 10-15 pairs in person. It also really helps when the exercises we prepared for them are purpose-built for pair-programming. When students can investigate and solve their own problems, then there is much less staff support required.

Brittany: We initially thought that online would be quite different because trouble shooting takes longer and so you would need more staff, especially for beginner's courses. But after our experiences over the pandemic and our online teaching in the Medical School I'd say that one staff member per 5-10 pairs is a good proportion in an online classroom. And sometimes we are constantly (virtually) running between breakout rooms, but there are times when we just roam, and 'poke our head in' to see how they're doing. It's really nice when students say "since you're here, check out this cool code we've written!". haha

Interviewer: And how are the sessions structured? Is there an introduction or demo at the start of the session? Or a goodbye at the end?

Brittany: It really depends what the task is, and how it is anchored in the course structure. We try to follow the flipped classroom format, where 'talking at students' is discouraged. And instead we try to use the precious time together to work on practical tasks, in pairs.

Some lecturers start with a mini-lecture or coding demo, where they show students in 10-20 minutes how to solve the first task or give any course updates, and then students are assigned to their pairs and start by retracing the lecturer's steps. Same with gathering everyone for a debrief at the end - sometimes we do it, but often pairs are so immersed in their work that we do not want to interrupt them.

Pawel: And in terms of structure, once students are introduced to this format, after a few weeks, it just happens quite effortlessly. The teacher is still needed for some logistics, like creating the pairs, break-out rooms. But overall it's sort of self-organising, like, you do not need anyone to be in charge. And that frees us, teachers, to focus on helping students. There's a book I love about those self-organising activities which can be used in a classroom or a meeting, it's called "Liberating Structures", check it out.

But coming back to the pair programming, a good structure goes a long way. Forgive me a harsh example, but once I had a small bike accident on my way to work, and when I finally arrived in the classroom, what I saw stunned me. My students had already, all by themselves, split into pairs and just started working away on their tasks! It was great, I had that fantastic feeling that you get as a teacher, that I am not really needed any more! That, once they understood the process, students just came to the sessions and get on with it.

Interviewer: Ok I understand the logistics a bit better now. And you

mentioned that some tasks are better than others for pair programming?

Pawel: What works best is doing practical things. Especially when the exercises are separated into a series of small-ish tasks. Like I mentioned before, working together doesn't work too well for passively absorbing content, because we all process at different speed and or encode knowledge differently. So it is all doing small tasks, and if any reading is required, we instruct students to do it together and discuss what they've read.

But pair programming is really most suitable for active learning. It works when you have to together agree on a way forward, or when you together to decide the next small step that will take us there. It's like escaping a labyrinth or solving a puzzle together. For our students this process usually involves doodling, discussion and eventually writing some code.

Interviewer: This sounds like something that would be useful when you are already quite advanced. Would it also work for beginners?

Brittany: Ha, it's actually really great for beginners! It can be new or uncomfortable the very first time, because we're just not used to this type of honest and transparent problem solving or vulnerability of learning with someone else. For people who are just starting to learn something, pair programming is a safe and kind way to develop their ideas, their vocabulary, and so on. They also learn how to communicate about a new topic effectively with another person. You know how we have active vocabulary, but also a passive vocabulary - words we would recognise, but would never use in a sentence.

When students switch who's driving, they have the opportunity to work on both sides of their understanding: actively saying things, and understanding someone who says things. It challenges their misconceptions and blind spots. They say that you only understand something if you can explain it, in pair programming you constantly have to explain things. Back to that medical rule of: see one, do one, teach one.

Interviewer: We spoke about it briefly before, but what about mixed experience groups, when some people are further on their journey than others. Would they clash? Do the more novice students get intimidated, or do more senior ones get bored?

Brittany: Like we mentioned before, it really is about following the structure and switching. It can be humbling for a more experienced student to have the role of navigator and active switching enables both groups of students in different ways and it helps them in different ways. In fact, I recently got feedback from a more advanced student that they found pair programming really useful because it forced them to slow down and actually think about the code, what it is doing, and why they were making certain decisions.

But also we live in an imperfect world and it does happen that one of the partners breaks the rules. Sometimes people hog the keyboard, or talk over the other person, or dictate to the driver what to type, even if it's not their time to type.

Pawel: You need to be on the lookout as a teacher, occasionally check on the pairs to see if everything works well. There are many small ways to create an inclusive environment, or to give students opportunities to recover their flow when something goes wrong. There's a chapter led by Charlotte in our book, which describes those recovery mechanisms (Desvages et al. 2026).

To create an equitable world with diversity and inclusion in focus, we can't just care for students who are ahead, maybe because they had a leg up in their previous education. Our main focus is not to just teach the best students to be better, but rather to open the opportunities that coding provides to everyone (Guest and Forbes 2024).

Interviewer: Oh, I see how pair programming could be a great practice for inclusivity, and how we have an important mission here as teachers to make sure that it is in fact inclusive.

Logistics (Room, Equipment)

Interviewer: I have a few more questions about the practical side of things. When doing pair programming in person, what equipment do you need? Will any classroom do? Do you need special screens, tables, chairs?

Brittany: You actually need very little equipment - one computer for each pair is enough. If we work in a computer lab, each pair uses one computer and passes the keyboard around. It also works with passing one laptop around, but laptops usually have smaller screens, and if a 'pair' has three students it gets a bit crowded.

In terms of the room setup, as teachers we need the ability to walk around the groups to check in on them and to help them if needed. Which is why row-by-row lecture theaters do not work so well, because you cannot work between the rows of chairs. You also need an ability for students to sit together and see what is on the screen, which tends to be difficult for those mobile chairs on wheels with little pop-up tables.

Pawel: My preference is a good, old-fashioned classroom with desks and chairs. But the technique is very flexible. We've seen groups of 3 students connect to classroom screens to see better. I've seen instructors who put students in larger groups of 5, passing the keyboard around, gathered around a big screen.

But keep in mind that not everyone can see and hear well, so to include everyone's voice we need to have a good setup. We teach students how to adjust their computers so everyone can see well, hear each other, and be comfortable.

Interviewer: You mentioned pair programming online, so is this something that you can do in online courses? Or even in hybrid courses with some students online, and some in person?

Brittany: Over the last decade we've tried every possible combination. During the pandemic we had to suddenly start teaching online, including with pair programming, and it turned out to be very manageable, and in some ways even easier to run. For

example, each group had its own breakout room, and the teacher could visit them with a click of a button. The students could also use the raise-hand feature to indicate they needed help. Instead of sharing a laptop, the driver would share their screen to show what they were doing.

In some way it made it easier to enact the roles, since there was no way the navigator would be tempted to touch the driver's keyboard from across an ocean. Among other benefits there's also the ease of switching between coding and doodling on a digital whiteboard (in-person students would doodle on pieces of paper). Overall, it was quite a manageable switch, with the student experience very similar to what we see in person. Now, years after the pandemic, we are still using pair programming in our online courses in the Edinburgh Medical School, with really good results!

Pawel: And indeed over the last few years in the Edinburgh Futures Institute we have been teaching large hybrid courses. Hybrid means that students in both modalities, some online and some in-person, pair-program together in real time. Imagine, a group of people in Edinburgh, sharing their screen and coding together with groups of people around the world. It was slightly chaotic, but also very inspiring and fun.

With my colleagues, Bea and Clare who wrote a book chapter about it (Alex, Llewellyn-MacRae, and Orzechowski 2026), we run our 30 person course in programming for Social Sciences like that. With our other colleagues I've run a similar course with 300 students. That's 300 people who would work in a hybrid way (some online, some in-person) with each other.

It requires a lot of prep, great knowledge of technology, and a really good team who can think on their feet. But it works! What makes it possible is the attention to small practical things like good earphones and microphones. When students can hear their partners without too much distraction in the background, all is well.

Interviewer: Wow, 300 students, half in person and half online, working together. What a thing to imagine! Amazing!

Tasks Students Work On

Interviewer: Ok, so it sounds like you're using pair programming a lot, in many different modalities. Tell me about the design of materials for pair programming sessions.

Brittany: What really works is if we have a set of small programming tasks. This way students can switch who's driving after each task. If tasks are larger, they could switch according to the clock (e.g. every 10 or 15 minutes). We try to have a few smaller warmup tasks at the beginning of the session, to give the pairs a bit of confidence. But once they're warmed up, the tasks can get more difficult and creative.

Interviewer: So in these tasks, what sort of code are you having the students work on? Do you give them a place to start, expected results, or tests? How detailed are the instructions?

Pawel: It really depends. In general we keep written instructions to minimum, since 'reading together' is not the best activity to do in a pair. Then we lean on the flexibility of pair programming, adjusting it to the level and topic of the course. For example, on a beginner course you may be given some code to get you started and more structure on what to do. Often you would get a worked example, maybe with something small to tweak in it.

On more advanced courses the tasks are often really open-ended, because students know how to navigate code, and also are used to working in pairs. The challenge is to provide enough guidance so that students are not lost, but also without making it a simple paint-by-numbers with no creative thinking or agency.

Interviewer: That makes sense. So I imagine each group will go at a different speed, choosing to focus on something else, or going deeper into another thing. What if they do not finish everything in the allocated session time?

Brittany: Oh, that is absolutely fine! Usually, it is the case that many pairs do not complete all tasks, or often they choose to skip some. But we don't let them get stressed out about it. Sample solutions are typically shared with students after the class, which can help to moderate the focus just getting a functioning answer without much thought or reflection.

Completing the tasks is not the main focus of the session, instead the real focus is on something else: to practice communicating and working with the human they are paired with. And developing a vocabulary for discussing new concepts and topics, and learning from each other. It does not matter how far they got into the materials, as long as they learn stuff really.

Creating Pairs

Interviewer: Ok, so what I'm getting here is that a lot of it hangs on the human relationship with your programming partner. How do you design the pairs? Do you do some kind of social engineering to create good combinations?

Brittany: It depends. We usually tell students who their pair will be at the beginning of the session. This makes things easier, and simplifies the chaos of 'where should I sit' and 'would you like to be in a pair with me'. And when creating those pairs we optimise for people having many different partners, preferably being with a new partner every time. This is great for building community, making friends, and the cross-pollination of skills.

But also for some people with social anxiety it can introduce uncertainty or worry, so there's always a way to tweak the pairs for those students or allow them to pick. Indeed our colleague Charlotte, from the math department, lets students pick their own partners so they can work with friends. From what Charlotte has shared, this can have a positive impact on attendance. It really depends what cohort of students you work with, and what are your constraints.

Pawel: And in terms of how exactly we come up with pairs... we use different strategies for that. Often it is just purely random, we shuffle a list of names and that decides who's with who. And

sometimes it's quite fancy, and we can be very thoughtful about creating the most diverse pairs. Since we're all geeks, we even wrote a website which creates perfect pairs for you![5].

One way or another, since pairs are generally not repeated, we lean on the gods of randomness to just sort things out: if you had a 'meh' partner this week, then chances are you'll have a better one next time. But also you can't engineer everything and that's ok. There's a lot of 'letting go' that you need to run a class like that.

Brittany: Yeah, above everything else, the pairing strategy needs to be practical and quick, for example on courses where attendance is patchy or many people are late... we just put people into random groups as they appear. It always works out in the end.

Interviewer: What if you have an odd number of students? How do you split 13 students into pairs?

Brittany: It's really not a problem! Often we create some pairs of 3 people, where they have two navigators and one driver. They still would use just one laptop but pass it between 3 people. This flexible definition of a 'pair' comes quite handy in unexpected moments: quite often we end up with 'pairs of 3' as a contingency plan, for example when someone is unprepared, or anxious, or takes time to warm up to the method.

Same goes for any logistical issues, like patchy internet in online classes, or someone being very late - a group of 3 is our generic solution to any trouble. Indeed, on some of my courses internet problems are so common that the groups have 3 people as default.

Pawel: But also our colleague Umberto in Psychology has been experimenting with groups of 5. Where one student is driving, and others take various other roles, like Scribe, Strategist, Researcher. In the industry this type of method is called 'mob programming', a little bit like with "pitchforks and torches" type of mob.

It works really well when the driver is pitching the solution to the group and they approve of it, or suggest improvements. It seems to work well, especially when well integrated into the design of the whole course. Pair programming is a very flexible way to collaborate, but it's important to communicate the roles, the structure and the expectations to students very clearly.

Running a Session

Before We Start / Preparing the Group

Interviewer: Right, that makes sense - the whole class should be on board and know exactly what they are doing. How do you achieve that? How do you onboard the class, especially the first few times?

Brittany: We start each course with explaining why the method works, and how to get the most out of it. Students should know what the roles are, which we sometimes illustrate with a live demonstration. It is also important that they know 'who is who' immediately - we tell them that the first thing to do once they find their partner is to designate the first driver. Within minutes they will switch anyway, and both will get to be the driver, so we want them to start coding as soon as they get into their pair.

Pawel: Yeah, since this method really requires students to follow the roles, the script - we ensure that only one person types, and that they only have 1 computer open in front of them. Whenever we check-up on a pair, we ask those questions, like "who's the driver right now?" and "when was the last time you switched drivers?". But also we make sure that if they get stuck, they know how to call for help, or get themselves unstuck (Zarb and Hughes 2015).

Interviewer: So what happens in the very moment when two students are being put together in a pair? What does the first 30 seconds look like?

Pawel: Just before sending students to their pairs we reiterate first things they should do in their pair, like a simple code of conduct. It's all common sense: how to greet each other; how to choose who drives first; and how to create a kind and open environment for everyone. But knowing where to start makes everything much smoother.

Brittany: It's important to remember that negotiating accessibility features is a part of that initial hello. Agreeing on things like text size and colour scheme, or whatever else is needed so that both partners can meaningfully and equally contribute.

Interviewer: I am imagining myself as I pass the keyboard to someone else. But this is my code, they will change my work, I imagine having feelings! Have you seen this sort of attachment to one's work in your students?

Brittany: Ha, that's a good one. What we've seen is people writing simpler code because they no longer write it for just themselves. What you write as the driver, has to be understood and 'accepted' by the navigator. The navigator needs to fully understand it, because in a minute they will continue from where you left off. A simple example is that if you use meaningful names for things, there is less confusion about what you meant, which also means less bugs in your code.

Pawel: It's quite eyeopening and creates a lot of moments of pride! You stop accepting quick, shoddy code... or just thinking "oh, I'll fix this later". Instead we switch to long-term, kind thinking like "I need to write this code clearly and simply enough, so when we switch drivers, the next person can continue without much drama".

[5] https://ddi-talent.shinyapps.io/pair-up/

Once Pairs Are at Work / Roles of Instructors and Students

Interviewer: And once the pairs are at work, what do the instructors do?

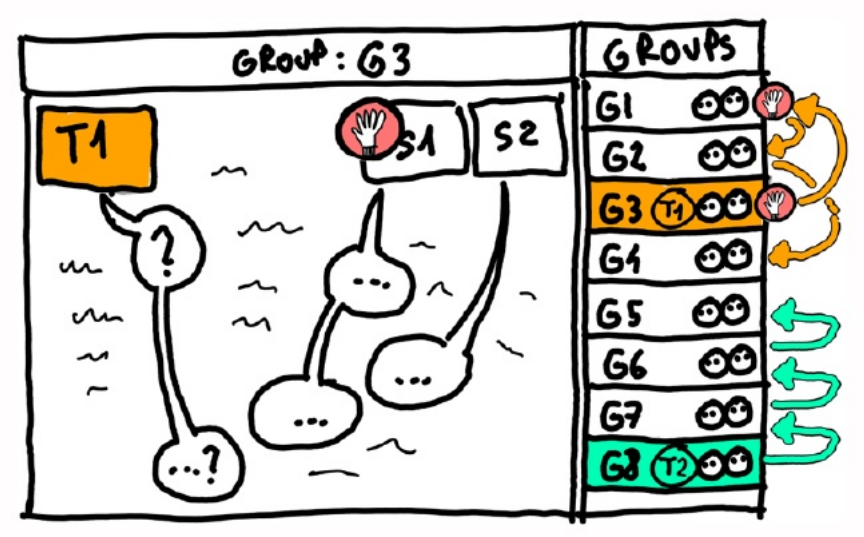


Figure 2.4: In online pair programming instructors continuously roam between breakout rooms, prioritising those with raised hands. Instructions take 'ownership' of parts of 'the room'. When instructor enters a breakout room, they first ask students to describe the problem and what was already tried. After that instructor will gives help students.

Brittany: We roam around the room and help the pairs. As a teaching team we take on two roles, really: being the facilitators and being the teachers. As facilitators we keep checking if students are doing the pair programming right: are they switching drivers, are they using just one computer, and such.

But really most of our time is spent being the teachers: checking if they are doing OK with the tasks, if they need any help getting unblocked or unstuck. Whenever someone raises their hand, we go and help them. And then once they are unstuck, we resume roaming. Usually in each class we would check in on each pair quite a few times.

Interviewer: So, as an instructor, when do you give help and how? Do you give your students the answers, or hints, or just ask leading questions? How stuck is too stuck? Do you ever just give them a leg up so they can start the next task?

Pawel: Totally, there's a subtle art of being a good tutor: knowing how and when to help. Not too early, but also not too later. Since the goal of class is to learn, not to just plow through the tasks, there's little value in just giving students the answer. It is ok to let students struggle a bit, because then they forge new ways to think.

But if it takes too long and is missing context, it can become frustration which is not good. It's sort of like lifting weights at the gym, you need some perseverance and effort, for things to stay in your head. We guide students and unblock them, usually just enough so that they can find the answer themselves. But we do help students a lot and in many ways, since not all struggles are good struggles.

As a tutor you tune into each individual situation and ask 'are we approaching a learning moment here, or are we just lost'. There are times when it's more beneficial to just unblock the pair, so they can move on to the next task. Especially when more learning waits for them over there.

Brittany: Getting unstuck or unblocked often includes asking students to explain what they have tried already and why they think it did not work. We also often switch to a whiteboard or a piece of paper to "doodle ourselves out of trouble". We try to use drawings to identify the solution first, and agree between partners what it could be. And only then they try to type the code which represents that doodled solution. At its core, solving problems is what we teach. Typing code is just the final step of that process.

Interviewer: So there is a lot of teacher skill there, but also a lot of students' self-regulation. For example, when should they decide that they are 'stuck enough' to ask for help? Do you teach students how they should approach problems?

Pawel: We do. Indeed, that's at the core of pair programming. Students try to solve the problem first with their partner before they ask the teacher for help. In primary schools kids are taught to escalate problems with this mnemonic "brain, book, buddy, boss". Your brain is your first point of call when you face a problem. Then your books and your buddies. Only once those failed you would approach the teacher.

When a tutor approaches a pair who asked for help, the first thing they would ask is "what did you try already?". That's important because solving things yourself creates learning opportunities. What's valuable here is that Brain, Book and Buddy is not a passive experience of asking or trying to remember something. We encourage students to actively experiment. In the context of programming, it means creating little code experiments, small 'sandbox' solutions, to tease out the answer. It is often faster than googling.

Brittany: Exactly! Running code and seeing what happens is the best way to go. I often tell my students "You won't break your computer, you might break the code but we can fix it". Let me give you a geeky example: two students are wondering what happens if you add two words to each other, like "pen" and "apple". Would Python just glue them making a "penapple" (all one word) or would it do what a human would and put a space in between them making a "pen apple" (as 2 words). Or maybe it will freak out and error! Students could spend 3 minutes or more googling for the solution, but they might also take 10 seconds writing a one line of code which adds "pen" and "apple" to each other, and just see what happens.

Pawel: Those tiny experiments are the very essence of what programming is.

Brittany: Indeed! Creating them does much more than just give you the answer - the process teaches you how to investigate problems, and how to understand the topics deeply. It's practicing the skill of problem decomposition - there's a saying that "there are no difficult problems in programming". If it seems difficult, you just need to decompose it into smaller and smaller problems, until they are simple to solve. (haha) and yes, it's that decomposition that is the real programming skill. And this is what students learn in pair programming.

Pawel: And coming back to the logistics of asking for help, students who need it would just raise their hand. Some classes use the Software Carpentries[6] model where students can stick a red or green post-it note to their screen, so teachers know how they are doing, and if they need help.

In big classes, like our EFI course with 300 students, our Teaching Assistants would act as the avant garde - they go to help students first. And if the problem is very challenging, then they escalate it further and call over the Lecturer. Really it is just more of that "brain book buddy boss" philosophy. To learn problem solving, as a whole class we need to practice problem solving.

End of the Session

Interviewer: I see. So the focus in a pair programming session is to get more practical experience. And how do the sessions end? Is there a structure similar to how you start the session?

Brittany: Sometimes, the time runs out and we all just go home. But it is also nice to have a moment at the end of the class where the pairs come back together to debrief. People share how it went and what they learned. It usually turns out that there were common issues and themes - it's good to feel this community vibe at the end (haha).

But quite often pairs are so deep into their work that they do not want to come back to debrief with the class. Students go "just 5 more minutes, we're doing so well!". And that's a great sign! Having too much fun in the classroom is never a bad thing.

Interviewer: Speaking of ending the pair programming sessions, I think we'll need to wrap up our chat soon as well. We've discussed a lot of fascinating stuff. I am intrigued and now I want to try Pair Programming in my own teaching. What's your one piece of advice for people like me, who want to try it in their classroom?

Brittany: I'd say: find a way to experience it by yourself. Since Pair Programming is such an experiential, immersive thing to do, you will really get the point of it once you have done it yourself. When you have had to drive, and navigate, and pass the keyboard to someone. Experience it to understand it.

All you'll need to start is: a willing colleague or friend; two hours of time; and a simple programming task (maybe a challenge from Tidy Tuesday[7], or Advent of Code[8]. You could even come to one of our taster sessions which we run at conferences and such, where we pair you up with another participant, we give you something to do and let you Pair Program! You can find instructions on how to run such a workshop by yourself on our website[9].

Pawel: And my biggest suggestion is, once you've tried it, go full steam ahead, without half-measures. It might be tempting to 'simplify' the method and remove some parts of it, but you might end up throwing out the "baby with the bathwater".

For example if you do not have clear roles (who's the driver, and who's the navigator) it might fall into chaos or into working separately. If you do not switch, the person with more confidence or experience can end up hogging the keyboard. Or if you create bigger groups, people might disengage and not take ownership of the exercise. I'd say, start with the simple 'vanilla' off-the-shelf pair programming, and then find what works best in your context.

Interviewer: This sounds like very useful advice and first steps! And if people want to know more, they can find out about it on the website of our community, pairprogramming.ed.ac.uk[10]! Thanks for talking to me today. I am Dr Kasia Banas, and I've been talking today with Dr Brittany Blankinship and Dr Pawel Orzechowski from the Edinburgh Medical School at the University of Edinburgh.

Brittany: Thanks for having us!

Pawel: Thank you!

---

[6] https://carpentries.org/

[7] https://github.com/rfordatascience/tidytuesday

[8] https://adventofcode.com

[9] https://github.com/teaching-programming/pair-programming-r-workshop

[10] https://pairprogramming.ed.ac.uk

References

Alex, Beatrice, Clare Llewellyn-MacRae, and Pawel Orzechowski. 2026. “Learning Together Across Modes: Online and on-Site Pair Programming in a Fusion Course.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Desvages, Charlotte, Brittany Blankinship, Umberto Noè, and Pawel Orzechowski. 2026. “Structured Group Work with Assigned Asymmetrical Roles and Switching: Lessons from Pair Programming Across Disciplines.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Goel, Sanjay, and Vanshi Kathuria. 2010. “A Novel Approach for Collaborative Pair Programming.” *Journal of Information Technology Education: Research* 9 (1): 183–96. https://doi.org/10.28945/1290.

Guest, Olivia, and Samuel H Forbes. 2024. “Teaching Coding Inclusively: If This, Then What?” *Tijdschrift Voor Genderstudies* 27 (2/3): 196–217.

Orzechowski, Pawel, Elaine Mowat, Karim Rivera Lares, Rebecca Sewell, Clare Llewellyn-MacRae, Bea Alex, and Kasia Banas. 2026. “3 Stars and 1 Wish: Small and Frequent Student Reflections Promote a Sense of Wonder and a Community of Vulnerability.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Zarb, Mark, and Janet Hughes. 2015. “Breaking the Communication Barrier: Guidelines to Aid Communication Within Pair Programming.” *Computer Science Education* 25 (2): 120–51. https://doi.org/10.1080/08993408.2015.1033125.

Structured Group Work With Assigned Asymmetrical Roles and Switching: Lessons From Pair Programming Across Disciplines

Charlotte Desvages

School of Mathematics and Maxwell Institute for the Mathematical Sciences, University of Edinburgh
charlotte.desvages@ed.ac.uk

Pawel Orzechowski

Edinburgh Medical School, University of Edinburgh
pawel.orzechowski@ed.ac.uk

Brittany Blankinship

Edinburgh Medical School, University of Edinburgh
b.blankinship@ed.ac.uk

Umberto Noè

School of Philosophy, Psychology and Language Sciences, University of Edinburgh
Umberto.Noe@ed.ac.uk

Introduction

In this chapter, we share our experiences with pair programming as a structured group work format that we believe fosters student engagement, well-being, and transferable skills. We argue that the same pedagogical benefits can also be realised when used for non-programming tasks.

What Is "Pair Programming"? (And Why Should You Care?)

In the software industry, people often work in pairs, which evolved over the years into a standard collaborative practice called "pair programming" (Williams 2010). Partners take turns playing two different roles: driver and navigator. The driver has exclusive control of the shared computer, and the navigator guides the driver by acting as their sounding board. Being in each role allows them to gain and contribute unique perspectives on the task at hand. Switching roles frequently allows both partners to get the most out of the experience.

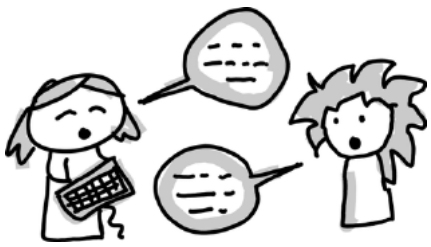


Figure 3.1: "I think your idea will work, let me type it in!"," I see a bracket missing on line 7", "Let's run this code and see what happens.", "Yay, we did it! It works!"

The purpose of the exercise is not only to produce better output (e.g., code) but also to: share knowledge and practice (cross-pollinate ideas); help each other to grow; create friendships and community; and foster a culture of being able to doubt and ask for help (you're never struggling alone).

In the professional world, pair programming benefits the product, the individuals, and the team. When implemented in the classroom, students can reap those same benefits, with evidence of positive impact on learning experience and outcomes (Hanks et al. 2011).

We are a team of UoE educators who have been using pair programming as a group work protocol in our teaching for several years. We have done so across many subject areas (psychology, mathematics, statistics, data science, medicine, epidemiology, business, EFI); levels of study (undergraduate and postgraduate); and delivery modalities (in person, online, hybrid). We have consistently seen the positive impact of this practice on our students and teaching teams.

In our view, the benefits come from the structured roles and the process, which are transferable to other production tasks beyond just writing code (Saltz and Heckman 2020). Indeed, any task where a group of people create an artifact (e.g., writing, weaving, drawing, problem-solving) could apply the same collaboration principles.

Structured Roles for Active Learning

Group work can be beneficial to learning, but navigating group dynamics requires effort, which can significantly worsen the cognitive load of learning something new. When roles are vague (or non-existent), communicating and negotiating responsibilities between group members can become an extra task in itself. For instance, when no-one in a team feels individually responsible, everyone might wait for the others to take initiative, and nothing gets done; or, one student might assume a leadership role, leaving others who are

less confident to follow passively or to disengage entirely.

As a structured way to work together (a "collaboration script" (Weinberger 2011)), pair programming gives groups a roadmap with prescribed roles for each student, and specific activities associated with each role. With the burden of inefficient group work lifted, students have more room to focus on learning. Furthermore, when clear roles are assigned, everyone has to take ownership of their part and actively contribute.

> "Making sure everyone takes turns being the driver means everyone needs to contribute." - Undergraduate Student, Psychology

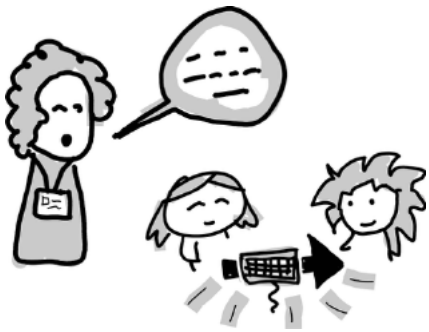


Figure 3.2: "Hello! Which of you is driving now?", "Is it time to switch drivers?", "Do you need help getting unblocked or are you just cracking on?"

As a result, we see students engage more. They take responsibility for their group, attend more sessions, and come better prepared to not let their teammates down.

> "Students were not engaging in the tutorial sessions at the beginning, which made them less useful. But it improved after having pair programming" - Postgraduate Student, Medical School

Having defined roles doesn't leave room for any one student to take over the task or speak over others, as can sometimes happen in unstructured group work. This is particularly important as many of our students have experienced marginalisation, and hence feel less safe putting themselves forward.

> "This is sort of just a problem with sexism, but I have really disliked group working in the past because I've been put with people (boys) who completely disrespect my work. This is why I'm opposed to the forced group working. It's better to work alone than be treated like that." - Undergraduate Student, Mathematics

Although the structured roles are not a magic bullet to prevent these issues, they can provide tools for both students and tutors to maintain a safer learning environment. One useful strategy is for tutors to casually check in with the groups, leading with the question: "who is the driver?". This does two things: first, it is a temperature check of whether the collaboration is working well; second, it reinforces the expectation that students should stick to the script at all times, even after the tutor leaves.



Figure 3.3: "What if I end up in a pair with someone who is mean, or much more advanced than me?", "I'm uncomfortable in socially complex situations. Will I know what to say and do in a group...", "Should I just always work with the same friend, or would I miss out on meeting new people, and learning new coding styles"

If a student is undermined by their partner, they can fall back on the script to reassert themselves as a valued contributor, or to communicate the issue with a tutor. This can reduce the emotional cost required to flag or confront bad behaviour directly, by giving both students and tutors a tool to defuse the situation.

Fostering Peer Learning and Community

Pair programming provides plenty of opportunities for peer learning. The driver has to verbalise their specific ideas and problems, discuss them with the navigator, and synthesise the outcomes of their discussion to "put things on paper" and advance the task. The navigator must be an active listener, observer, and helper, and can take the leading role in higher-level strategic thinking. Both students are exposed to each other's approach, and learn from each other by cross-pollination.

> "The most useful part of [pair programming] is being able to collaborate with other classmates, which allows me to discover aspects I hadn't noticed in my own work through the questions raised by others." - Postgraduate Student, Mathematics

> [A core learning outcome from the course was] Experience in pair programming and trying to explain concepts to another student. Sometimes it's quite easy to "see" how you'd do something yourself, but actually explaining this instead of doing it is more difficult, so the online sessions were very useful to practice this . I also found it very valuable to see how others work and how they understand code differently." - Postgraduate Student, Medical School

Students also appreciate how discussing with a peer can generate new ideas. A pair programming environment is highly conducive to discussion, creating a free flow of ideas where the outcome is more than the sum of its parts. Since only one person can write things down, students must verbally negotiate a shared understanding of the agreed way forward.

> "We have different ideas, and co-operate with each other could generate more ideas." -

> Undergraduate Student, Psychology
>
> "Group discussions often have unexpected results." - Undergraduate Student, Psychology

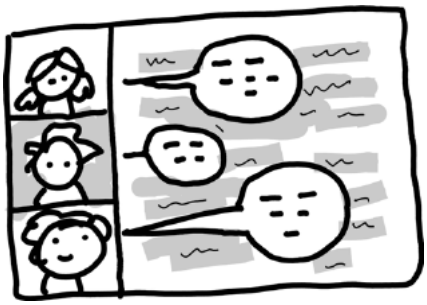


Figure 3.4: "So the way I understand it, is ...", "I see another way to solve this!", "That's cool! I never thought about it this way"

Although the roles are asymmetrical at any particular point in time, taking turns ensures that all students benefit equitably from playing both roles over the course of a session or a semester.

> "The practice of being a driver and a navigator is very useful. These two different roles provide me with distinct perspectives on the code. When I'm the driver, I focus on how to complete the code, and when I'm the navigator, I gain insights into how my fellow students understand the code, which helps me learn new things." - Postgraduate Student, Mathematics

Beyond direct benefits in the classroom, we have also seen pair programming help build a sense of community among students and enrich their personal life at university. Students build a network of peers that they can turn to for support later. This is especially critical in distance learning, where students do not have the opportunity to naturally meet peers (e.g., outside of class).

> "The group work is very helpful. Not only have I made new friends, but I now also have people that I can rely on when I need help. Group work has also made the individual workload seem much lighter and we get things done faster." - Undergraduate Student, Psychology
>
> "[It] was nice to get some 1-1 interaction through the pair programming in what could have been an otherwise solitary course." - Postgraduate Student, Mathematics (delivered online in 2020/21)

There are many ways to assign students into pairs, which might influence the classroom experience and community building in different ways. For example, many students are more likely to attend class if they can work with the same friend each time, but this might mean that they miss out on meeting new people. Swapping partners each week will create a network of student-to-student connections which contributes to cohort building. Finally, stretching the definition of "pair" programming to groups of 3 or more (with a single driver) can also work well. Allowing for more navigators can create flexibility in terms of logistics (odd number of students), technological problems (connection issues in online teaching), student preferences, or accessibility.

> "It gives you the perfect opportunity to work with other people and make potential new friends." - Undergraduate Student, Psychology
>
> "[about using PP on 4 different courses] Thank you for the interactive pair programming sessions [across the academic year], they made a big difference in creating a community of online learners." - Postgraduate Student, Medical School

Tackling Issues as a Community of Practice

Many of our students have experience of dysfunctional group work, where different group members have different priorities, expectations, or skills. Pair programming can exacerbate this issue, because each student is expected to contribute actively to the joint work, whether they are driver or navigator. Inevitably, pairs will occasionally be incompatible, in a way which can undermine the effectiveness of peer learning. For instance, a very wide gap in technical proficiency or level of preparation can make for a frustrating experience for both students, leading to disengagement and even resentment.

> "I fully agree with the idea of group working, but in practise it usually results in people being left behind by those who understand better [...] and unfortunately in maths way too many people are mean and make the beginners feel really bad." - Undergraduate Student, Mathematics

Although the strategies discussed above can minimise this, there is no single correct solution. Whatever way you decide to implement groupwork, you will encounter issues, and your approach will need to be adapted to your discipline, classroom, or even cohort.

We have come together as a community of practice for mutual support. We have found it incredibly valuable as educators to discuss experiences, tackle challenges, and share examples of good practice.

This Chapter Was Pair-Programmed

Pair programming has worked in our programming classes, but we very much see the value of this kind of structured group work outwith a programming context. For instance, we "pair-programmed" the writing and editing of this very chapter over two sunny afternoons. We took turns being the driver typing on a laptop, while everyone else navigated, contributing their ideas and wording. We believe that the principles we have discussed are widely applicable across disciplines, and we look forward to finding out how colleagues adapt it in their classrooms.

References

Hanks, Brian, Sue Fitzgerald, Renée McCauley, Laurie Murphy, and Carol Zander. 2011. "Pair Programming in Education: A Literature Review." *Computer Science Education* 21 (2): 135–73. https://doi.org/10.1080/08993408.2011.579808.

Saltz, Jeffrey, and Robert Heckman. 2020. "Using Structured Pair Activities in a Distributed Online Breakout Room." *Online Learning* 24 (1): 227–44. https://doi.org/10.24059/olj.v24i1.1632.

Weinberger, Armin. 2011. "Principles of Transactive Computer-Supported Collaboration Scripts." *Nordic Journal of Digital Literacy* 6 (3): 189–202. https://doi.org/10.18261/issn1891-943x-2011-03-06.

Williams, Laurie A. 2010. "Pair Programming." *Encyclopedia of Software Engineering* 2: 651–59. https://doi.org/10.1081/e-ese-120044181.

Pair Programming in Practice: A PhD Tutor's Reflections

Sarah Elliot

Usher Institute, University of Edinburgh
s.s.elliot@sms.ed.ac.uk

From Student to Teacher

I am a PhD candidate in healthcare data science, having learnt R and Python programming at university. I vividly remember my own experiences as a beginner: persistently feeling lost or behind my peers due to my non-coding background, jargon-filled error messages, and overwhelming online solutions. Alongside these frustrations, I also recall the satisfaction of mastering a new skill and the sense of achievement that followed solving a difficult problem. These memories shaped my approach to tutoring during my PhD. I was motivated to support students in acquiring technical skills, and, importantly, build confidence while stepping into a new programming space that can often feel intimidating.

I worked as a tutor on an introductory R course designed for beginners. Students enrolled with a wide range of experience levels: those with no technical background, individuals transitioning to R from another programming language, and those with some self-taught experience seeking a more structured workflow. The course was delivered online on a part-time basis, with most students working full-time as well as studying. This course format increased the accessibility for learners who might otherwise be excluded from traditional full-time study, but it also presented pedagogical challenges. Students were geographically dispersed, had additional personal and professional commitments, and had different technical home set-ups. Learning primarily took place through scheduled online sessions, online discussion boards and independent work, and opportunities for informal peer interactions were more limited. Therefore, the course coordinators utilised learning activities that supported both technical development and social connection — notably, pair programming.

Pair Programming Builds Confidence

The emergence of generative artificial intelligence (AI) has reshaped how beginner programmers can approach coding tasks. **Problems that previously required trial-and-error troubleshooting can be resolved within seconds of requesting an AI solution.** These AI tools are valuable but can reduce the understanding of the resulting code. Learning to program involves more than obtaining functional code: it is important to understand core concepts such as data and its structure, functions, good coding practices, problem solving, and transparent interpretation and communication of results. The best way to develop these skills is through actively engaging with problems as opposed to passively receiving an answer (Bonwell and Eison 1991; Chattopadhyay, Ryan, and Pockrandt 2022).

Pair programming is a strategy that prioritises learning the process of coding as opposed to just the end product. It involves at least two individuals working together on the same task at the same time, with one 'driver' who types the code and thinks out loud and one 'navigator' who instructs the driver, reflects on the code, and looks out for mistakes (Bryant, Romero, and Boulay 2008). Working collaboratively, students articulate their reasoning, suggest alternative approaches and interpret challenges together.

The confidence-building effects of this approach were evident across the students. More experienced students derived satisfaction from supporting their peers and consolidating their own understanding, while less experienced students gained practical experience in a low-risk setting. It is possible that students felt less anxious contributing ideas in small group settings as opposed to whole-class discussions (McDonough 2004). Observing the learners' engagement over time, the students were seen to become more confident undertaking both roles, with

less hesitation in sharing screens and contributing ideas. Coding alongside peers who were also in the process of developing their skills was valuable as explanations were drawn from recently experienced challenges and solutions, as opposed to second-nature expertise from a formal teacher. Challenges were faced with collective resilience as opposed to feeling like an individual struggle. Pairs actively gave and received feedback, providing learning opportunities that are missed during individual tasks (Storch and Wigglesworth 2006). Content and skills learned collaboratively, and practised during pair programming, seemed to be retained for longer and utilised in future work and assessments (Shehadeh 2011).

Due to having a relatively small cohort and strong staff-to-student ratio, the progress of individual students was visible to the teaching team during this course. I observed students who regularly engaged with pair programming benefit from confident engagement with the course overall. The hands-on programming gave them more coding practice than teacher-fronted activities, and the growing confidence in their community and their abilities resulted in willingness to ask questions and share partial solutions on the online discussion boards. Collaboratively solving challenges, communicating throughout learning, and becoming confident with uncertainty emerged as valuable skills developed from pair programming.

There was also a social benefit to pair programming. Due to running online, many students had few opportunities to interact with their peers outside of the scheduled sessions. Reflections from students highlighted that paired or grouped activities created opportunities for connection, helping students feel less isolated. Pair programming supported learning and community building, which can be difficult to achieve in a remote learning environment (Laal and Ghodsi 2012).

Supporting Each Student To Make Mixed Pairings Work

Implementing pair programming was not without difficulties. Grouping students with widely varying experience levels required close consideration. **Mixed-ability pairings are productive, but it is important to monitor the contributions of each student.** The experience gap should support collaborative learning and skill development, particularly among those paired with a more-proficient peer, without the less-experienced participant becoming too intimidated to contribute or the more-experienced participant feeling insufficiently challenged (Shehadeh 2011; Maguire et al. 2014).

Technical constraints also impacted participation. Due to the remote delivery and global reach of this course, it was common for some learners to have unstable internet connectivity or other technical issues that limited their ability to share their screen and take on the 'driver' role. This influenced the roles that individuals could choose and the group dynamics. This was improved by using groups of three where needed, allowing more flexibility in chosen roles.

Although collaborative learning is a valuable skill, it can be undermined if students see their classmates as competitors (Schinske and Tanner 2014). In this course, students are explicitly told that their final grades would not be in competition with their peers, and this framing helped encourage collaboration throughout pair programming. Throughout the pair coding tasks, students shared ideas, experiences, useful tools and innovative functions, which appeared to improve the quality of the learning and understanding of key concepts across the cohort.

A Richer Learning Experience for Both Students and Tutors

While facilitating pair programming, a tutor's role is to support students as they work through tasks and navigate challenges. This involves allowing students to attempt challenges before offering help and, when needed, providing helpful hints as opposed to full solutions, balancing guidance with autonomy. The nature of pair programming means students must verbalise their thought process, making it easier for tutors to assess the learners' needs and identify any misunderstandings. While the class is working in pairs, tutors have the opportunity to assist individual learners or pairs (Soleimani, Modirkhamene, and Sadeghi 2017). As more experienced R programmers, tutors can share practical, real-world approaches for students to explore. Tutors are able to observe pair dynamics to learn which pairs work well and who needs more support, to inform future groupings. Tutors should lead by example to support student learning: this includes communicating ideas clearly, modelling good coding practices, maintaining approachability (such as keeping cameras on in remote sessions), and asking reflective questions.

Reflecting on this experience, using pair programming to teach non-programmers resulted in a way of both building confidence in technical skills and fostering community in a remote environment. In an era where AI can readily generate solutions, pair programming allowed students to gain hands-on coding experience and programmatic thinking, while also developing transferable skills such as problem solving, communication, and collaborative learning. **As a PhD tutor, pair programming allowed me to share my own strategies during these sessions and also learn new approaches from the students, showing that learning is richest when multiple perspectives are shared at every level.**

References

Bonwell, Charles C., and James A. Eison. 1991. “Active Learning: Creating Excitement in the Classroom. 1991 ASHE-ERIC Higher Education Reports.” ERIC Clearinghouse on Higher Education, The George Washington University, One Dupont Circle, Suite 630, Washington, DC 20036-1183. https://eric.ed.gov/?id=ED336049.

Bryant, Sallyann, Pablo Romero, and Benedict du Boulay. 2008. “Pair Programming and the Mysterious Role of the Navigator.” *International Journal of Human-Computer Studies*, Collaborative and social aspects of computer software development, 66 (7): 519–29. https://doi.org/10.1016/j.ijhcs.2007.03.005.

Chattopadhyay, Ankur, Drew Ryan, and James Pockrandt. 2022. “Scaffolded Live Coding: A Hybrid Pedagogical Approach for Enhanced Teaching of Coding Skills.” In *2022 IEEE Frontiers in Education Conference (FIE)*, 1–9. IEEE. https://doi.org/10.1109/FIE56618.2022.9962513.

Laal, Marjan, and Seyed Mohammad Ghodsi. 2012. “Benefits of Collaborative Learning.” *Procedia - Social and Behavioral Sciences*, World conference on learning, teaching & administration - 2011, 31 (January): 486–90. https://doi.org/10.1016/j.sbspro.2011.12.091.

Maguire, Phil, Rebecca Maguire, Philip Hyland, and Patrick Marshall. 2014. “Enhancing Collaborative Learning Using Pair Programming: Who Benefits?” *All Ireland Journal of Higher Education* 6 (2). https://doi.org/10.62707/aishej.v6i2.141.

McDonough, Kim. 2004. “Learner-Learner Interaction During Pair and Small Group Activities in a Thai EFL Context.” *System: An International Journal of Educational Technology and Applied Linguistics* 32 (2): 207–24. https://doi.org/10.1016/j.system.2004.01.003.

Schinske, Jeffrey, and Kimberly Tanner. 2014. “Teaching More by Grading Less (or Differently).” *CBE Life Sciences Education* 13 (2): 159–66. https://doi.org/10.1187/cbe.cbe-14-03-0054.

Shehadeh, Ali. 2011. “Effects and Student Perceptions of Collaborative Writing in L2.” *Journal of Second Language Writing* 20 (4): 286–305. https://doi.org/10.1016/j.jslw.2011.05.010.

Soleimani, Maryam, Sima Modirkhamene, and Karim Sadeghi. 2017. “Peer-Mediated Vs. Individual Writing: Measuring Fluency, Complexity, and Accuracy in Writing.” *Innovation in Language Learning and Teaching* 11 (1): 86–100. https://doi.org/10.1080/17501229.2015.1043915.

Storch, Neomy, and Gillian Wigglesworth. 2006. “Writing Tasks:: The Effects of Collaboration.” In *Investigating Tasks in Formal Language Learning*, 157–77. Multilingual Matters. https://doi.org/10.21832/9781853599286-011.

Brain Book Buddy Boss Break: A Practical Debugging Framework for Learners

Mar Doig

Engineering Education Lead at Perk

Introduction: The Debugging Mountain & The Error Misconception

Harken your memory back to your first error message, however many years ago that might be. You likely felt a small jolt of panic, a worry you'd irreparably **broken something**. Nowadays, you hopefully see errors as the lane bumpers of your programming journey. Errors help us understand flaws in our original design, consequences of code changes on original code, and limitations of our system.

When teaching students at the beginning of their journey, it's important to remember they will require psychological coaching to reframe these errors from failure, to progress. As experts in their field, they rarely make mistakes in their work. In learning a new skill it's key to foster the growth mindset that's so fundamental in programming.

This growth mindset muscle must be strong, so strong in fact that it can conquer the usual process of fixing one error only to find another one. An experienced programmer would see this as progress, the stepping stones that will lead to their desired end result. Whereas students often see it as compounding failure, eroding away at their frustration tolerance. This can be a huge blocker in their learning and lead to demotivation or the classic "programming is just not for me" mindset.

This chapter will lay out the **5Bs framework**[11] **- Brain, Book, Buddy, Boss, Break -** a guide for students encountering debugging. This strategy codifies the organic path of problem solving that is cultivated in engineering. It will allow the student to develop a systematic approach to problem-solving that will keep them moving forward in their debugging, increasing their frustration tolerance as a result. Throughout this framework, we'll also explore how learners can thoughtfully leverage modern AI assistants as powerful aids, while still ensuring they build deep conceptual understanding and robust problem-solving skills.

A Note About Debugging and AI

However you feel about the broad usage of AI, their integration into learning environments demands our careful consideration. These powerful assistants can remarkably accelerate experienced developers and are invaluable for getting the job done efficiently. However, for the novice programmer focused on building foundational understanding and their own problem-solving abilities, they present a double-edged sword. If an AI consistently serves the perfectly cooked meal, the learner may miss the crucial, sometimes messy, process of learning to prepare it themselves. This may hinder the development of their core analytical and debugging muscles. With this in mind, the 5Bs framework presented in this chapter will explore how to thoughtfully integrate AI at each stage, aiming to strike a crucial balance: leveraging these incredible tools appropriately while always prioritizing and fostering deep, independent learning and critical thinking.

[11] We expand on the "Brain Book Buddy Boss" technique, which is used in classrooms, but seldom mentioned in teaching literature or books. We could not track down its exact origins, but in chapter 4 of Claxton (2018) it is introduced as a variant of "Try Three Before Me". "Try Three Before Me" guides pupils who got stuck to first try solving or troubleshooting their problem themselves in 3 different ways, before they escalate it to the teacher. "Brain Book Buddy Boss" is a more specific variant, which suggests specific stops on that troubleshooting journey.

BRAIN: Engage Your Mind Before Your Keyboard

This core step cannot be understated. Before engaging with any resource a student must build practice in thinking about their problem. They will likely be used to, from their current problem solving ability, creating a hypothesis and testing it to discover a result. It's likely this is their approach when writing code too. The key is to remind them it's the first port of call for an error message too.

In this part of the process, no AI should be used. The thinking process is precisely where the learning happens, and frustration tolerance is built. The struggle between the student and the problem will exercise their problem-solving muscle thanks to the resistance put up by the code. AI could give them the information straightaway, but this is not the goal.

Here are some practical techniques a student can engage:

1. **Read the error message:** This is a skill in itself. Knowing how to spot which file and line are causing the offending problem will give us a first place to start the investigation, and may even reveal it was all down to a simple misspelling or syntax error.
2. **Reproduce the error:** Does the bug happen if you run it again? The classic turn it off and on again still yields results.
3. **Identify the gap:** Note what you are getting and compare against what you were expecting. Is there an interesting relationship between these two pieces of data? Is the answer "off by one"?
4. **Formulate a hypothesis:** Turn the gap into a hypothesis, "I think X is happening because of Y."

At this point we are ready to test Y until we figure it out, or we find a new error and go back to the beginning.

5. **Simplify and isolate:** Look at what changed since the code worked. Try to isolate the problem and see if it's still reproducible. You can also comment out all code and uncomment it out slowly until the error reproduces, finding the source of your problems.
6. **Sample inputs and testing:** Change or hardcode a variable to test your hypothesis. This is where print debugging enters the scene too. Print key variable states before and after suspected problem areas, and remember to label them!
7. **Debugging:** Use debugging tools built-in to the IDE to set breakpoints, and step into functions.

This process is designed to be done in loops, until one moves forward or has to use another step in the framework. Encourage students to keep checkpointed versions of their work, as undo commands only provide a limited safety net.

Tip for Educators

Encourage the student's self-reflection during this process by prompting them with simple questions:

- What have you tried so far?
- What does the error say?
- What happens when you change this?

BOOK: Consult Your Knowledge Base

Official sources of knowledge should be the core of a student's debugging. These are presented in the order the student should consider consulting them.

Own Materials

A common first step is for the student to consult their learning materials, their notes or their code. This is usually a very useful resource, as they are deeply engaged with it and it's likely to have similar examples to the problem they're trying to solve. Students may be motivated to produce tidy, well-commented code in the course of their learning, when they know they're likely to go back to it when they're stuck.

Documentation

Despite it being a complex source to parse, I would always encourage the students to engage a primary source, the documentation. Getting comfortable reading technical documents is part of the learning process, and in most cases that's where the most authoritative answers will be found. Finding answers to a problem via reading documentation is a hard skill to master, but it is extremely valuable.

Books

While it isn't the quickest way to solve a bug, many engineers' favourite pathway into deeper understanding is reading published titles. Authors have taken much time and deliberation over every sentence in their book, and this leads to the highest quality explanations of concepts. Books are a form of linear learning, which contrast to the other options' fragmented learning. While a google search might fix a bug, reading a book chapter will fix the misunderstanding that caused it in the first place.

Reading around a topic when one is stuck can be a natural way to gain deep understanding without feeling like one must set aside time for learning & development. By using these casual

opportunities to learn, the student can create a lifelong habit of continuous learning.

Googling

Of course many students would straight-away google the error. This is appropriate in many situations, and often gets the student unstuck. However, it can lead to the copy paste trap where the student layers Stack Overflow answer upon answer, until their code works. Even if this leads to working code, it's likely a messy approach, and it's possible that the student doesn't fully understand the why of their solution. It's also likely that they cannot explain why this code is the most appropriate solution for their problem. A great standard to set for students is to set out the clear guidance that they must be able to explain the what and why of every line of code they submit.

Artificial Intelligence

Nowadays of course, it is more likely that a student will put their error or whole code into an AI chatbot. This can be a great option if handled appropriately, as the students can engage in conversation about the error, leading to deeper understanding. A curious and engaged student will enhance their learning through the conversation, but the pitfalls are obvious. A student who is looking to simply submit an assignment on time is likely to skim through the explanation and take the correct answer.

A great way for students to leverage AI is by using it to demystify some of the harder-to-parse sources, like documentation. The student can paste the part of the documentation that they believe relates to their issue and ask the AI to clarify or simplify the language.

As explained earlier as it relates to AI, the limitations of this technology should be made clear to students. It can, and indeed does, confidently hallucinate answers and provide references that do not exist. Students should verify the knowledge passed on by AI with external sources, especially if the AI's suggestions start to sound implausible or make the code worse.

Tip for Educators

Provide your students with an AI prompt that they can put into their chatbot which asks the agent not to give the students any code, and instead act as a Socratic Teacher who will engage in conversation about the topics and guide the student towards their understanding. For example:

```
I am a student learning Python. I am currently stuck on an error and I want to practice my debugging skills. Please act as a Socratic tutor. Do NOT give me the corrected code. Instead, ask me leading questions about my logic, point out specific lines where my mental model might be mismatched with the syntax, and help me arrive at the solution myself.
```

BUDDY: The Power of Collaboration

The most incredible thing about the buddy step, is that it doesn't require a buddy at all. Many programmers are extremely familiar with solving their own problem halfway through explaining it to a colleague! So, let's help students embrace the power of articulating their problem.

Rubber Ducking

A very common practice for software engineers is rubber ducking. This is when someone takes a rubber duck that is sitting on their desk for this express purpose, and one explains their code to them. In the process of performing this explanation, one usually sees the gap in the logic, or the misunderstood assumption, and has at least one new approach they can try to get themselves unstuck. While the rubber duck is a common talisman, this explanation can be done to anything: your cat, your favourite lego figure, or even out loud to yourself! My personal favourite that generates less public concern when you're among colleagues is to write down the explanation of the logic on paper as if I were speaking it out loud.

Artificial Intelligence

Of course, as mentioned in the section above, if a student chooses to use AI they should primarily use them as a Socratic tutor who they can have a discussion with. For me, this places AI firmly in the category of Buddy. Using the prompt above, the student should be able to engage with the AI as they would a colleague, with the great advantage of its 24/7 availability.

Fellow Humans

The two prior options described are great resources for the middle of the night bugs. However, if there is an option to discuss with an actual person, I would recommend this option above all.

Explaining your reasoning to someone else holds you to a higher standard of use of technical language. Whenever

you describe your code to someone else, you organise your thoughts before you speak and you take cues from them to expand on explaining the parts that are most important. Their intuition collaborates with yours in real time to create better understanding. You'd be surprised how often discussing something with a colleague will also enhance their understanding of concepts too.

Finally, learning is a collaborative endeavour: it becomes more effective and more enjoyable when shared.

Tip for Educators

Provide as many opportunities as you can to engage your students' technical collaboration skills. A great way to do this is by encouraging pair programming as part of tackling coding challenges.

Pair programming is an excellent practice for a myriad of reasons, (and rightly deserves a book in itself!), but for the scope of this chapter, its main advantage is that it develops the student's ability to explain their thinking using technical language. Students will collaborate in their problem solving and get each other unstuck during the pair programming session, leading to higher understanding and higher morale.

BOSS: When to Escalate

In my experience, students will err on the side of waiting too long to reach out, much to their detriment. A student's fear of seeming like they've not understood something they believe they should understand by now, or their pride in wanting to reach a solution by themselves, stops them from reaching out to their tutor. Time is a non-renewable learning resource and we must make this clear to our students.

Helping them build out the skill of reaching out to a superior at this early stage of their career will prove invaluable in a professional environment too, when not reaching out won't just be an impediment to their problem solving but possibly to their career advancement too.

A common first question is "how long do I wait?", and the reply is that it depends on the length of time you have to complete the task. If a student has been set a task in class that's 30 minutes long, and 10 minutes in they're still in the beginning stages, then it's appropriate to reach out. If they're working on an assignment that they have weeks to complete, they ought to try for longer than 10 minutes before reaching out.

Tip for Educators

The hardest part of reaching out can be the intimidation of knowing, ironically, that they've probably now waited longer than they should have. To encourage the students to reach out when appropriate and push through that feeling, it can be helpful to give them a framework of preparation they can use as a guide to feel more comfortable in escalating.

- **Context** - What are you trying to do? What part of what you're tacking is new to you? What specific error message or bug are you getting?
- **Brain** - What is your understanding of the problem? What hypotheses have you tested?
- **Book** - Where did you go to get more information? Was it helpful? Did it confuse you further? Do you have any similar sample code which you do understand?
- **Buddy** - Have you talked it through with a colleague? Did you discuss this problem with AI? Where in their explanations did you get lost?
- **Question** - What one specific question could your tutor answer you that would lead to the most significant unblocking?

Ensure you treat a student's escalation with utmost care, they are likely building a fragile muscle which will require tackling one's own ego and some old stubborn habits.

BREAK: The Essential Reset

Much like the ego that gets in the way of escalating, a similar feeling can get in the way of the key step of taking a break. Switching from the focused mode of active problem solving to diffuse mode, where problems are processed in the background. A 5 minute break to drink a glass of water or stare into the distance out the window is often much more productive than insisting on going through 5 more minutes of debugging.

The break step can, and arguably should, go in between any of the previous steps outlined. It should not be a last resource when the student has been at a problem uninterruptedly for an hour. Any time that the current step is making things less clear instead of more clear, Break should be utilised. As an instructor, I set out breaks during the lesson to model this approach.

It's also important to note that while short breaks should be part of their workflow, long breaks should be part of their life. It's our responsibility as educators to give our students a holistic perspective on learning, which incorporates time away from screens entirely. In the specific context of software bootcamps for example, students put a lot of pressure on themselves to continuously code and improve their skills. It's key to remind them that in order to learn well the next day, they will need some time in their day where they are doing something that refills their energy tank like eating well, exercising, and sleeping.

Conclusion

This framework codifies what is a natural, iterative loop for experienced problem solvers. By sharing the 5Bs mnemonic, we provide students with more than just a debugging checklist: we give them a toolkit for student resilience and a way to navigate debugging without falling into demotivation.

As educators, our role is to consistently remind them of the core principles that underpin this framework:

- **Errors are the Path to Progress** - We must move students away from seeing errors as failure and toward seeing them as the lane bumpers of their journey.
- **Time Management Practice** - We must emphasize that time is a non-renewable learning resource. Students should be taught that escalating to a Boss is an act of professional efficiency, not a sign of weakness.
- **The Power of the Reset** - We must model the Break as a technical requirement. Stepping away allows the brain to defocus, which is often when the most elegant solutions surface.
- **Thoughtful AI Integration** - By using AI as a buddy or a documentation translator, students leverage modern tools without sacrificing their journey of understanding

The 5Bs are a useful way to remember the common steps in problem solving, but the goal is learning to live in the loop.

References

Claxton, Guy. 2018. *The Learning Power Approach: Teaching Learners to Teach Themselves (the Learning Power Series)*. Crown House Publishing Ltd.

A Guide on Empowering Students to Develop a Coding Mindset

Neil Bray

UK Health Security Agency
neil.bray@ukhsa.gov.uk

Samantha Alvarez Madrazo

School of Public Health, Imperial College London
s.alvarez-madrazo@imperial.ac.uk

Lucia Michielin

Edinburgh Futures Institute, University of Edinburgh
lucia.michielin@ed.ac.uk

Nick Jayanth

School of Public Health, Imperial College London
nick.jayanth19@imperial.ac.uk

Sam Skipsey

School of Physics and Astronomy, University of Glasgow
Samuel.Skipsey@glasgow.ac.uk

About Us

We're writing this chapter because when we started teaching students to code "from scratch", we would have liked something similar to show us what's worked for other people. Much of the existing resources are direct teaching materials, rather than discussions of how to teach, and, especially, how to teach in the service of other skills, not as an end unto itself. A "coding mindset" develops over time as a combination of problem decomposition, embracing iterative improvement, and expectation management (each of which are also applicable outside coding!). Rather than a rigid set of rules, all the examples below can be seen as "good enough" practices we developed across time, aiming to support beginners in developing those aspects of themselves.

It's also important to recognise that everyone's situation will be different. The five of us are predominantly from statistical or quantitative disciplines (all but one!). This inevitably means that we tend to think of coding as a means to perform statistical analysis and therefore the "coding mindset" we want overlaps with a statistician's. We also mostly tend to "think" in R or Python (one of us thinks in Julia), which will almost certainly flavour our examples. We hope this chapter will be of use to you, it is however, not a reflection on your own practice if some of our specific examples don't work for you. After all, there are as many legitimate use cases for coding as there are people and disciplines. If you can adapt the concepts here to teaching engineers to code in Ada, or game designers to code in Lua, or any other combination, then more power to you.

Set Learning Expectations

> Reassure learners that the goal is not to become an expert coder overnight but to achieve clear and realistic learning outcomes. Encourage learners to ask questions and to not compare themselves against more experienced coders.

Many students may be attending a coding class for the first time, but exposure to more experienced coders can create unhelpful comparisons. We discuss the role of the educator in setting realistic expectations early in the course, which allows learners to engage more fully with the learning process.

The learner may be concerned that they should already know some syntax and that others will be more experienced. It's important to reassure the group that the course will start from the basics and that the class will be paced for all learners, not just those with prior experience. Beginner coders are often more reluctant to ask questions and should be reassured that all questions are welcome. Students may be intimidated by the complexity of a computer language, so it can be helpful to clarify that the goal isn't to learn everything. Instead, the focus will be on core concepts such as the structure of the language, for example, functions, loops, and if/else blocks, which will be taught as part of the course. Sharing clear learning objectives at this stage can help the learner understand the achievable scope of the course. Giving learners a longer-term view by explaining that even experienced coders tend to focus on the areas that they use the most can help put this into perspective.

It's also useful to explain that memorising code comes with repetition, which is largely outside the scope of introductory courses and is not an indicator of aptitude. Many students will keep snippets of code for future reference, review documentation, use large language models, and

perform web searches to find syntax as needed. Reassuring students that these are normal practices, even amongst experienced coders, can help them focus more on understanding how the code works rather than trying to memorise every detail.

With the availability of code on the internet, it is important to encourage learners to question why the code works rather than simply accepting that it does. Developing the ability to critically appraise code found online depends on a strong understanding of the programming language itself. Educators can support this by encouraging learners to explain their code and understand how each part contributes to the overall project.

By focusing on what the learning expectations are (and aren't), educators can help learners focus fully on adopting a coding mindset, a skill that transfers across computer languages.

Building Motivation

Demonstrate coding solutions to inspire the class. Assess motivation for learning how to code by exploring confidence and perceived importance. Use this to support learning and reassure the class that everyone has similar concerns and aspirations.

Understanding and improving learners' motivation is a useful preparatory step in building a coding mindset. We suggest that this can be achieved through a combination of code demonstrations, assessing how confident learners feel about learning to code and how important they believe it is.

A learner starting their coding journey may become inspired to learn by seeing an example of a coding solution that has direct relevance to their area of interest. If the learner's main area of work is producing reports, then demonstrating how data can be transformed into a HTML document containing attractive tables and charts will grab their attention. If the benefits of a coding approach are further highlighted, such as time savings and reproducibility, then this can build motivation even further.

Concepts used to assess motivation are fully explained by Miller and Rollnick (2013). Here, we provide an exercise that focuses on exploring the learners' sense of confidence and importance that they hold in relation to learning to code. For this exercise, online tools provide a way to collect information from the class, provide anonymity, and allow other learners to see each other's responses. Learners should be reassured that there are no wrong answers to this exercise, and it is designed to support their learning journey.

Learners are first asked to rate how confident they feel in learning how to code on a scale of 1 to 10. This is then followed up with two questions: why they didn't choose a lower number to uncover strengths, and why they didn't choose a higher number to identify barriers. This process is then repeated with learners rating how important they feel learning to code is, again giving reasons why they didn't choose a lower or higher number.

A learner may respond that they rate themselves as 5 out of 10 in terms of how confident they feel in learning to code. The learner explains that they have some prior experience writing code, but that they became frustrated by frequent error messages. However, they rate the importance of learning to code as 9 out of 10 because they understand it would be beneficial for their career. In response to this, the educator can offer reassurance that frustration is one of the common concerns that students face when first learning to code, as explained later in the chapter.

We suggest this exercise only needs to be completed once as an assessment tool and is most effective in the early stages of a learning programme, during the first class if possible. Educators benefit from this as it provides insight into the needs of the class. The need for additional support, such as drop-in workshops, can be assessed, which is especially helpful in remote settings with limited contact time. As responses are shared within the group, for learners, it can create a sense of peer support when others share similar concerns, and it may also highlight positives they hadn't yet thought of. It is anticipated that this exercise will help learners embrace the learning process rather than being undermined by self-doubt.

Embracing Feedback

Computer error messages and receiving human feedback are a normal part of coding. Learners should embrace both, reflect, and share their learning.

Coding is an iterative process where improvements are made with each version. This is often prompted through feedback via code reviews with other learners or educators, or by computer error messages. These forms of feedback are valuable teaching tools, and embracing such feedback is key to effective code writing.

For beginners, computer error messages can feel overwhelming and may be interpreted as a sign that they are unable to code. Educators can help reassure learners that these messages can be seen in the same light as any other type of feedback. The challenge is to encourage learners to stay positive and view the process not as repeated failure, but as testing different approaches before arriving at a working solution. One advantage of computer error messages is that they provide immediate feedback and contribute to the learning process as the student learns what works. Human feedback builds on this by offering advice on how to overcome errors and make improvements to code that is already functioning. For example, a code review with peers can highlight future issues with code which may not be immediately apparent.

To help learners receive and process both types of feedback, reflective learning provides a supportive space for learners to think through the error messages they've encountered. The purpose of the exercise is to help learners normalise the experience of overcoming error messages rather than seeing them as a negative critique of their ability. A structured template can help learners describe the situation and the steps taken to rectify the problem. The suggestion for the template is as below, although it can be amended to suit different groups as required; an example is given using R. The template focuses on the coding aspects of resolving errors. Guidance on how educators can support students in managing the frustration that often arises is outlined later in the chapter.

Example: Template To Reflect on Error Messages

- **Describe the scenario:** I was practising filtering a dataframe using the built-in `iris` dataset in R.
- **Describe the problem encountered:** the dataframe did not filter.
- **Which part of the code was the problem?:** `filter(iris, Species == "setosa")`
- **What was the error message?:** `Error: object 'Species' not found`
- **How the problem was solved:** I realised I needed to load the `dplyr` package to access the function that I wanted to use; therefore, at the start of the script, I entered `library(dplyr)` to load the function.
- **What further guidance is needed:** I wasn't sure why it didn't tell me that the function itself wasn't loaded.
- **Lessons or code snippets to take forward for next time:** Always check to ensure necessary packages are loaded.
- **Comments from other learners or educators:** `filter()` is also a function in the `stats` package, which is loaded by default. This is why the error message didn't say the `filter()` function could not be found.

Learners are invited to pick an error message they have encountered over the course and present it using the above template in a short, five-minute presentation. By sharing their reflections with the class, learners can see different problem-solving approaches, resolve error messages, practice receiving feedback, and develop resilience.

Normalising Frustration

> Frustration is a normal aspect of coding for beginner and experienced coders. Include examples of sources of frustration into your teaching methods and provide support whilst the learner works through them.

There is no way around it: writing and building code is a frustrating activity. No matter how many years of experience you have, there is a high chance that the first time you are going to run your code, you are going to get an error. A task that you think would be done in 10 minutes can end up taking hours, and you may encounter error after error.

Teaching coding effectively, particularly to novices, presents unique challenges that often centre on the students' emotional reactions to the difficulties they encounter. Experienced practitioners understand that dealing with frustrations and obstacles is part of the learning process in coding. However, newcomers often find it hard to adopt this mindset. This frustration tends to intensify when students contrast the smooth demonstrations of code in class with their own stumbling attempts to solve exercises or apply concepts to different datasets.

As educators, it is crucial to adopt strategies that help students navigate these frustrations constructively. One such approach involves helping students to draw parallels between their coding struggles and challenges they have faced while learning other new skills or hobbies. Encouraging students to reflect on previous learning experiences where they felt similar frustrations can be enlightening. Discussing how they addressed these challenges and eventually overcame them can provide valuable lessons that are applicable to learning to code.

Introducing intentional small errors or typos in demonstration code can also be an effective teaching tool. This strategy not only makes classroom sessions more interactive but also provides opportunities to address how to manage and rectify errors in coding. It is important for the instructor to remember where these errors have been placed to ensure smooth handling during lessons. Highlighting these errors during demonstrations serves as a powerful reminder that the polished code often shown in classrooms is the product of much iteration and failure.

Organising follow-up classes, particularly those that adopt a flexible, open-ended format, can be exceedingly beneficial for students. These sessions, which we tend to call BYOD ("Bring Your Own Data") classes, provide a unique opportunity for students to apply the concepts they have learnt in a practical, supportive environment. Such classes encourage students to bring their own datasets or project ideas, which they can then explore with the guidance of an instructor. This hands-on approach not only helps to consolidate their learning but also enhances their ability to apply coding skills to real-world scenarios. Having an instructor present to offer assistance is crucial, especially when students encounter obstacles. This support can significantly boost their confidence and competence, enabling them to tackle challenges more effectively and with less apprehension. During these sessions, it is particularly productive to encourage collaborative problem-solving. For example, when a student brings a specific dataset or a coding issue to class, working as a group to resolve these issues can be extremely advantageous. Not only does the student who brought the data benefit from collective insights, but other participants also gain a deeper

understanding of how to approach similar problems. This type of interaction often leads to valuable discussions that reveal multiple approaches to solving the same problem. The group overall benefits from this by enhancing their analytical skills and fostering a more comprehensive understanding of the subject matter.

Framing Coding as a Problem-Solving Activity

Instruct learners to outline the coding problem they are trying to solve, break this down into manageable steps and write code to address each one.

Viewing coding as a problem-solving activity offers a two-fold approach. First, defining the problem to be solved; and second, problem decomposition: breaking the problem down into smaller problems and outlining the steps needed to solve it. This allows learners to focus on the overall process rather than feeling discouraged by perceived gaps in their current skills. This idea can be introduced to learners by asking them to consider the decisions that they make in their own lives, to choose one, and to break it down step by step.

An example to demonstrate this is shown below. A student may decide to arrange a day out with friends to a local attraction and break the planning down in the following way:

1. Research what attractions exist locally
2. Arrange a date and time with friends
3. Book tickets
4. Make a meal reservation
5. Arrange travel once times are decided

When tasks are broken down in such a way, they appear more manageable, thereby reducing cognitive load. Just as planning a day out can be broken into manageable steps, coding tasks can be approached in a similar way. This process, called pseudocoding, involves outlining the steps in plain language before writing any actual code. For example, if a learner is asked to produce a graph showing the total number of items sold by week during the summer of 2024, they might first ask themselves, "What needs to happen to produce the graph?" After some consideration, they may list the following steps:

1. Load data
2. Filter the data so that it only includes data for the 2024 summer period
3. Count how many items were sold by week
4. Produce a bar chart to show items sold by week
5. Add a chart title and axis labels
6. Save the chart

With this list in mind, the student can then ask more targeted questions about how to go about writing the code. For step 1, the student may ask themselves, "How do I load a `.csv` file?" They can then begin answering this question by researching code. They then continue through the steps until they have a fully developed workflow.

It is anticipated that this approach will help learners think through a coding problem and identify the steps they need to take irrespective of their coding ability. As learners gain experience, developing this discipline will support their ability to write more complex and structured code.

Myth Busting

Learners may arrive at classes with self-limiting beliefs, educators can help replace these with more constructive thoughts to help aid the learning process.

Common self-limiting beliefs that students might hold often fall into the themes of the learning process, self-belief, writing code, and the relevance of coding to them. To address these, we have compiled a "Myth vs. Reality" table summarising what we believe to be the most common misconceptions. The goal is for students to identify any myths they may hold and feel encouraged to replace them with more constructive thoughts.

Where possible, the educator can encourage students to add to the table with their own barriers, either by using those highlighted in the motivational exercise described earlier in the chapter or following a brainstorming exercise. It is important that educators use supportive language when working through this exercise. For example, rather than describing code as a "failure", they can take the opportunity to reinforce that coding is an iterative discipline. This helps develop the coding mindset and prepare learners for working on their own projects outside the classroom.

This activity empowers learners to tackle their self-limiting beliefs and also provides educators with insight into their students' mindsets.

Myth vs. Reality: Common Misconceptions Surrounding Learning to Code

Myth	Reality
You need to know everything about coding for it to be valuable.	Most coders specialise in the areas they work in, and employers value the ability to teach yourself new technology more than trying to know everything.
You need to learn a specific language.	During your career you may use different languages depending on what your focus is. The important part of learning to code is to think like a coder, as this mindset is transferable across languages.
You need to know the syntax by heart to be a good coder.	If you know what to search for online, you are already halfway to finding a solution.
If you have tried to code before and failed, you won't be able to learn to code.	Failing is part of the process of coding, and it usually leads you to new ways of improving the code
You will be surrounded by people who are better coders and will get left behind.	Everyone has their own journey when learning to code. You will learn faster by asking questions and collaborating with others.
You can code better if you do it on your own.	Sometimes you may want to code alone; however, coding with others gives you access to benefits such as feedback, support, and technical help.
Coding is purely mathematical and if you are not good at maths, you will never be good at coding.	Viewing coding as a problem-solving activity and writing the steps with words can help you to understand that it is not purely mathematical.
Your code should work on the first try.	You will spend a lot of your coding time fixing errors and debugging. It is unrealistic to think you will get it completely right on the first try.
You need to work in technical roles such as web services or statistics for coding to be relevant.	Coding is not limited to one type of role. You can use coding for developing websites, games, managing databases, creating applications and generating artwork.
You can learn the basics, but advanced skills such as modelling will be too difficult.	If you pursue a data modelling career, you will spend a lot of time focusing on data quality rather than complex modelling. If you learn the basics, it helps to make the advanced skills more achievable.

Know Your Audience

Consider the needs of your learners and modify your teaching methods accordingly. Understanding specific needs and using relevant examples encourages learners to engage with the learning process.

The authors of this chapter represent a diverse set of fields, and demonstrate below how understanding their audience and tailoring teaching accordingly, helps to enhance the learning experience.

Public Health (Samantha and Nick)

In the online Masters of Public Health, students come from a wide range of backgrounds and stages in their career, some of whom have little familiarity with coding. Feedback from alumni suggested that students required a stronger foundation in coding prior to starting the core module in Statistics for Public Health. This crucial feedback helped us as educators to develop and deliver the R installation session in the term prior to students starting the module. The session includes teaching students to identify their R version, basic coding, identifying different data types and initial data exploration using one of the built-in datasets. Additionally, we strongly suggest students who are not confident in R programming to take an online R basics course. The session is recorded so students who cannot attend live can watch it later, but often we have found that being in person can alleviate any coding related anxiety. Students who are apprehensive about coding find this session very useful and engaging, and it gives them the opportunity to build a relationship with us educators. This gives them a confidence boost which we see developing during the course of the module. The students regularly attend live sessions and office hours, and benefit from the support, whilst evolving a resilient and resourceful coding mindset.

Public Health (Neil)

I've led and supported introductory coding sessions to public health and wider healthcare professionals who are typically new to programming and want to extend their analytical skills. They are often experienced analysts who want to transition from a spreadsheet-based workflow towards a reproducible analytical pipeline. Their current processes are often time consuming, consisting of repetitive manual steps that can make quality assurance difficult. A common task facing analysts is report production and this is often used as the focus during training sessions. During the session, I emphasise the parallels between their current processes, for example, rather than producing a table manually, this can be created and formatted using code. As time is often short in the workplace to learn new skills, a worked example showing how raw data is transformed into tables, plots, and commentary and then combined into a report serves as a template that learners can adapt with their own data.

The session ends with a call to action for learners to identify one area of their work to replicate using code, helping build momentum towards incorporating coding more extensively into their workflows.

Arts, Humanities, and Social Sciences (Lucia)

The training programme I lead is specifically designed for researchers from the Arts, Humanities, and Social Sciences who are keen to integrate digital and computational methods into their research. The courses are elective and non-credit bearing, but, given that their time is limited, the content must be directly relevant to their research interests. Attendees come from diverse academic backgrounds and frequently encounter significant challenges in understanding and using computer science terminology. Consequently, it is essential to minimise the use of jargon and to focus on fostering discussions that facilitate the understanding and application of the methods being taught. This approach not only aids in demystifying complex concepts but also supports participants in relating these new tools to their existing research frameworks. For further insights into the pedagogical approach and design of the programme, see the chapter "Interdisciplinary by Design: The Centre for Data, Culture, & Society Training Programme"[12].

Physics (Sam)

Teaching coding to Physicists, we want to emphasise the connection to physical problem solving — but also give them a "concrete" output. So, our lab explicitly front-loads enough material to generate images, and gives the students control over presentation aspects such as colour choice. Writing some (relatively simple) code that makes a complex image (we use fractals, of course) helps to amplify the feeling of achievement, versus just printing some text to the screen.

We also find that explicitly linking the problems solved to the material students encounter in other courses they've taken recently — or are taking concurrently — helps. Showing how you can solve, say, problems in astronomy that were actually worked by historical astronomers, but in seconds rather than months, is inspiring to students. Astronomy also gives more opportunities for that key visualisation aspect; we can animate planets moving around the sun and so on.

When we begin teaching C, the first compiled language the students have met, we explicitly begin by giving them broken code. This normalises the process of trying to compile it, reading what the compiler says is wrong, and iterating on this. This helps to break down the "fear" of compiler error messages — we know the code is broken, so the framing is that it is helping us spot the mistakes, not berating us.

[12] C22_interdisciplinary-by-design.qmd

Developing the Coding Mindset

Coding is as much about mindset as it is about syntax. We hope this chapter has shown that building confidence, resilience, and collaboration matters just as much as technical skills. Remember: every coder faces frustration, but these challenges are stepping stones, not barriers.

However you choose to teach or learn, focus on progress rather than perfection. Support each other, ask questions, and celebrate small wins. The right mindset will take you further than any individual language or tool. Wishing you, and your learners, a rewarding coding journey ahead.

References

Miller, William R., and Stephen Rollnick. 2013. *Motivational Interviewing: Helping People Change*. 3rd ed. New York: The Guilford Press.

It Depends: How to Develop Judgement in Programming

John Wilson

Edinburgh Medical School, University of Edinburgh
jwilso3@ed.ac.uk

Brittany Blankinship

Edinburgh Medical School, University of Edinburgh
b.blankinship@ed.ac.uk

Introduction

Programming courses often focus on what to write... syntax, functions, packages, patterns, pipelines. What gets less attention, even in "intermediate" or "advanced" courses, is how context influences the coding approach and how to dynamically adjust approach to fit the context. Programming courses and tutorials usually give advice on writing DRY, clean, expressive code. We believe there is a gap in how such advice is presented that can be filled by considering three tactics that in combination create a programming "mindset space". Having an awareness of these tactics allows a learner to consider which tactics are most useful at a given time, for a given problem, and to understand they can change in response to the different pressures their current situation might be exerting. As a way of making this explicit and teachable, [13] the three tactics we suggest are being Independent, Conscious, and Social (Figure 7.1). And further, we refer to these tactics as "forces" because they can be used to push against, or accommodate, the pressures felt as appropriate to the immediate context.

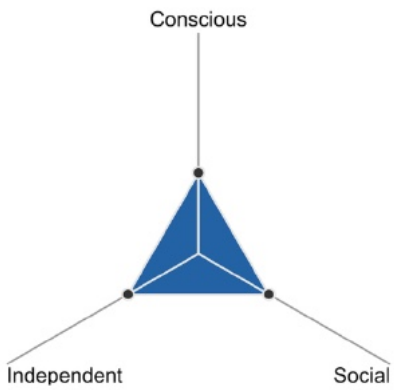


Figure 7.1: Radar plot of coding forces visualized with the 3 forces being equally balanced

The key idea is simple: good programming is not a single fixed standard, regardless of what language you might be programming in. Instead, different situations call for different combinations of skills, habits, approaches, and considerations. The classic "well, it depends..." applies to coding as well! This is to say, that in each project or task, the appropriate intensity of each coding force may vary.

The 3 Coding Forces

Independent coding focuses on problem-solving autonomy and self-sufficiency. This coding force is all about developing coding approaches without hand holding. This often presents in debugging, especially knowing when and how to navigate documentation or help files, search web resources (e.g., StackOverflow, books, blogs, etc.) effectively to find workable solutions and crucially adapt them to your context. Independent coding is about daring to try things out or experiment with ideas, adapt existing approaches, or develop your own new approaches without step-by-step guidance from a blog, colleague, or teacher.

Much of personal learning lives here. Think "I can do it all by myself, and how to implement it (all by myself), and know where to look for help if not". When primarily leaning into or applying the independent coding force you might produce exploratory scripts, small experiments, and the kind of trial-and-error work that helps to build competence and intuition.

Strong independent coding develops with time and once solid, presents as the confidence to be able to independently learn new packages or

[13] Here we have decided on a radar plot to represent the coding forces. It is not absolutely perfect, but this is a limitation we are accepting for now for clarity of communication. A Venn diagram seems like a good way to present these the forces, however, this gives an impression of three discrete "modes" of programming. The radar plot is more accurate in that it can show varying degrees of agency along each axis. A ternary plot we feel would be even more precise because it also acknowledges that there is a trade-off inherent in moving along any single axis. But ternary plot can be difficult to interpret, so we have spared you this.

workflows. Independence is what allows programmers to keep moving forward when they encounter unfamiliar problems. This is one of the hallmarks of "strong coders". It is not about knowing everything but having the confidence and skillset to learn and apply (nearly) anything, or to proactively decide to seek help when necessary. Programming skillsets like problem decomposition and debugging come into play here, which are built and strengthened through practice and exposure.

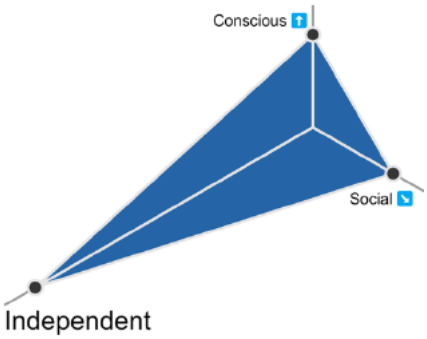


Figure 7.2: A dominant Independent force. Being close to the centre of the radar plot suggests zero independence, waiting to be given answers. The far end of the axis suggests taking the initiative to make progress, and judging when, how, and where to find necessary help.

Conscious Coding focuses on awareness and intention. This coding force is all about being mindful and making deliberate (and informed) choices about the structure, workflow, tools you choose to use in your code base. Conscious coding requires thinking ahead about structure, assumptions, and trade-offs rather than just making code "work". When coding consciously, you need to consider the bigger picture of how and where your code will be implemented, how it can be maintained, and why you are writing a given bit of code, sometimes even at the level of individual lines. When this force is the dominant force in your mindset, code comments should document why certain decisions were made (e.g., using one suite of packages over another due to downstream integration, etc.).

This is where questions about workflow, reproducibility, and tool choice live, including if or how to use AI tools! [14] Always bear in mind what AI produces, when it is helpful, and when reliance on it may hinder learning, flexibility, or long-term understanding. This is especially important when you are developing your skills. Learners may well encounter experienced coders that routinely use AI, but they should consider that perhaps those coders have already mastered the skills they are trying to learn! One good piece of advice is to type every single line of code AI produces with your own fingertips - avoid the temptation of mindless copy-pasting! This force is not about speed in making code that "works" or simply executes, it's about making conscious decisions using learned judgement.

Conscious coding asks you to slow down and reflect:

- Do I understand what this code is doing?
- Do I understand how to use this solution in another situation?
- Is this approach appropriate for this task?

If the answer is no to any of the above, take the time to experiment and learn (moving into the independent coding force) until the answers are yes. Otherwise, if time does not permit exploratory learning, do not use an approach or solution you do not understand.

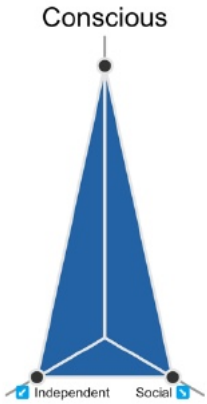


Figure 7.3: A dominant conscious force. Being close to the centre represents a lack of consideration about the code being written, such as the blind/mindless copy pasting of AI outputs. The far end of the axis represents making decisions based on long term maintenance of the code, only implementing solutions you understand and can debug.

Social Coding focuses on communication. Code is written to be read by other humans and executed by the machine. Social coding recognizes that most code is read (by other humans) more often than it is written. In this way, code itself is a form of communication, which you should want to be as clear as possible. We do not code in binary (can you imagine how gross this would be) for a reason!

This coding force is all about writing code that other human-beings can read and understand quickly and easily as well as reuse and trust. This often presents itself in robust documentation and ensuring consistency in procedures and approaches in a given script. Doing this well means writing code with clarity of meaning - i.e., consistent code style, descriptive naming conventions, and clear documentation through judicious use of comments, and properly documenting your functions.

When coding with a focus on the social force, your code may become longer, taking up more lines - which is not a bad thing! It does not mean your code is not DRY (the principle of Do not Repeat Yourself) but rather balances such principles with readability and

[14] Here we intentionally use AI generally as an umbrella term including not just Generative AI or LLMs, but AI developing tools like agents.

communication. Collaborative projects, research codebases, (and... assessed work in courses...), where maintainability and clarity are as important as correctness, depend heavily on a strong application of the social force. A useful realization for a learner to have is that collaborators may be people that you never have an actual face to face conversation with. Socially aware code, then, has to be friendly and approachable and the best ways to do this are to provide clear, accessible documentation, and code that is also self-documenting.

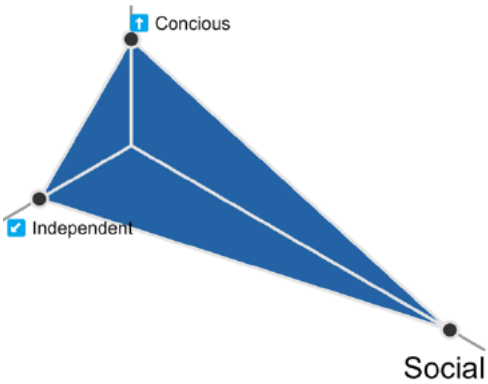


Figure 7.4: A dominant Social force. Being close to the centre represents, possibly rapidly developed, code that is ugly or without structure. The far end of the axis represents code nicely refactored for maximal readability, with the intention of sharing with collaborators or colleagues.

Different tasks call for the forces to be applied with different intensities, to actively balance them in the way that best fits the situation. What that balance might look like, is a deliberate choice of the coder:

- **quick exploratory work** may mean a dominance of independence because this allows faster iteration and experimentation.
- **short-lived scripts** might justifiably ignore social expectations. If that exploratory work transitions into being **integrated into a proper project with collaborators**, the influence of the conscious and social forces will become more important, likely requiring a tidying up and reconsideration of the code.
- **collaborative projects** need more from conscious and social forces
- **troubleshooting** may require a shared dominance of the conscious and independent coding forces
- **research and work projects** typically sit near the centre, requiring independence to solve problems, consciousness to make good design decisions, and social awareness so the rest of your team can understand, reuse, and replicate the work.

As a Tool for Reflection – Questions To Ask Yourself in Navigating These Coding Forces

We have intentionally chosen the term "force" to emphasise that these are not discrete modes you switch between, but influences that are always present... and which you have agency over. Each force can be applied with varying intensity depending on the situation, which is where your agency comes in.

This framing makes these forces something you can actively work with. They become levers you can pull to adjust each force, to manipulate how strongly each one shows up in your coding workflow. In practice, this isn't about turning one force "on" and another "off", but more like pulling levers that allow you to fiddle with the relative intensities of each force. This leads to a continuous tuning or rebalancing of forces as required by the current context.

Of course, those adjustments do not happen in a vacuum. They are often shaped by "pressures" - internal or external signals such as deadlines, collaborator expectations, or even just that weird feeling that there's something not quite right in your code. The radar plot reflects this idea. Rather than separating forces into neat regions it shows them all present at once, just at different intensities. The resulting shape is simply a snapshot of how things are balanced at a given moment (Figure 7.5).

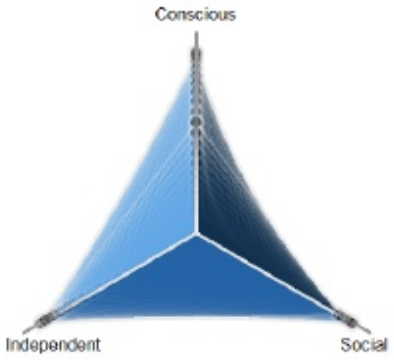


Figure 7.5: Ever changing balance of independent, conscious, and social code forces. In the online version, this is a dynamic gif but presented here as a static snapshot.

These three forces define a "mindset space", different regions of which are better suited to different applications. Moving about this mindset space requires reflection and awareness. This can be regularly asking yourself, "What am I doing right now and what mindset do I need to adopt?".

Importantly, it should be understood that using these coding forces is not a matter of switching from one to another. It's not as if you turn off "social coding" and suddenly become a purely independent coding machine. You do not take a metaphorical turn off Conscious Drive in order to exclusively be on Independence Avenue. Rather, you deliberately and dynamically alter the relative intensities of each force adapting to the context or circumstance.

You can think of the radar plot like a 3-way seesaw with each coding force functioning as a lever arm of the seesaw. All three forces are always in play just as all lever arms are in play on a seesaw though with different weights. As a visual, this analogy is useful because it highlights that raising one lever arm affects at least one of the others. In practice, this feels less like stepping between modes and more like

constantly rebalancing, or repositioning. You might lean heavily into independence when poking at a new problem, then gradually bring in more conscious structure as you converge to something worth keeping and then add social clarity as things settle down or need to be shared.

Speaking of our own tendencies, it is often that we (and possibly also the case for you) naturally fall into fast code that is not social or conscious. Code is never perfect the first time it is written, which is to say, we never get it perfect right out of the gates. Outputting final "socially acceptable" code is iterative, where each iteration changes the intensity of each force. When designing course content, however, the social coding force strongly dominates in the first instance.

Finding the Right Balance

Some questions you can ask yourself when searching for the right balance between these coding forces:

To increase the influence of the Independent coding force:

• I am stuck, how can I create a mini-reprex[15] for this issue?

To increase the influence of the Social coding force

• Will a collaborator (or marker) understand this code?

To increase the influence of the Conscious coding force

• Where or how will this code base be used? Does it need to be over-engineered and consider all edge cases, or no?

• Oh, I have a lot of functions, should I maybe separate them out into a separate script and source them in for debugging and maintainability?

More generally, when building up the awareness of the relative dominance of the coding forces and developing the reflexivity skill, it can be useful to take a step back and reflect on the following:

- What mindset and coding force am I operating in right now?
- What mindset should I be in right now?
- How do I even know what mindset I am/should be in right now? What is the primary purpose of the task I am working on?

Pressures and Rebalancing

Importantly, these coding forces are not separate "modes" that you switch between like changing tools in a toolbar or dramatically entering a new mental state while cinematic music plays in the background. All three forces are always present to some extent. What changes is the relative intensity of each force and how heavily you choose to lean into them depending on the situation.

This is why we use the term "force". A force is something that can be applied with varying strength. You can increase the intensity of independent coding when experimenting with a new idea or debugging an unfamiliar issue. You can increase the conscious force when making decisions about structure, reproducibility, or long-term maintainability. You can increase the social force when preparing code for collaborators, assessment, or future-you three weeks from now wondering what on earth past-you was thinking.

The important point is that you have agency in how these forces are balanced. Different situations create different pressures, and they may suggest a certain balance over others. So, rephrased, the context may influence the intensity of each force, but it is you who actually decides how heavily to lean into each force.

Some pressures are **external**. Deadlines may push you (temporarily) away from careful, consciously structured code, toward rapid exploratory work. Collaborator expectations may increase the need for clarity or documentation or consistency. Assessment tasks often increase the importance of social coding because your marker is, in a maybe slightly awkward sense, one of your collaborators.

Other pressures are **internal**. You may notice that your code is becoming difficult to debug, difficult to explain, or increasingly fragile as your work becomes more complex. You may realise you are over-engineering a tiny script that somehow now contains five layers of wrappers and several coding strategies named after medieval architecture. These are all signals that your current balance of coding forces may no longer fit the task particularly well.

With experience, programmers become better at recognising these pressures and adjusting accordingly. This is less about following fixed rules encountered on the world wild web, and more about developing judgement about when to increase or relax the emphasis on a particular force. You might begin with very independent exploratory work, gradually introduce more conscious structure as ideas settle, and later increase the social force once the code needs to be shared, reused, or maintained.

The radar plot reflects this idea. It is not intended to divide coding into neat regions or categories, but to show a changing balance of forces at a given moment. The resulting shape is simply a snapshot of which forces are currently being emphasized and how strongly they are being applied. Thinking back to our seesaw metaphor, the radar plot is a snapshot of the seesaw at a given moment. The goal is not to maximise every individual axis independently, but to arrive at a balance appropriate for

[15] What is a reprex? It's a reproducible example, as coined by Romain Francois. The goal of a reprex is to package your problematic code in such a way that other people can run it and feel your pain. Then, hopefully, they can provide a solution and put you out of your misery. The vast majority of the time creating an excellent reprex reveals the source of your problem. It is amazing how often the process of writing up a self-contained and minimal example allows you to answer your own question.

the task at hand. The important part is not any single axis alone, but the overall shape created by how the forces interact. As such, the plot is best understood as a snapshot of the relationship between the forces, not simply three separate values.

Moving Into the Next Dimension

Thus far, the framework as we have presented it can safely and easily live in a 2D mental space, represented by the radar plot and seesaw metaphor.

If you will permit us to further extend this metaphor, imagine that as you gain more awareness and experience on the seesaw of coding forces that the lever arms no longer move up and down but become a 3D puzzle pieces. As a learner acquiring a new programming language, you can think of your progression through the learning journey and the coding forces as a process of acquiring the puzzle pieces. Once an understanding of the independent, conscious, and social pieces are acquired, you realize they are complimentary elements that fit together and form a globe-like mindset-space. If you think of each force as a direction of travel within that globe, a new action is unlocked: spinning the globe and navigating the full 3D space!



Figure 7.6: Extending the 2D radar plot into a third dimension lets us imagine a continuous volume, through which we can navigate by changing the influence of the coding forces, until we find the region that corresponds to the best balance between the forces.

In this metaphor, meta-awareness and reflection is the glue that fits the puzzle pieces together forming the globe and allowing you to take on the hands of God role and move the globe. This meta-awareness is only possible to reach once you have knowledge of all 3 forces. It is required to then apply a fitting intensity and direct forces that feels appropriate for the problem at hand.

Using the radar plot imagery when applied to a specific problem it is unrealistic (and difficult to visualize) maxed out intensity applied to all 3 forces. After all, as human attention is a finite resource. This is where the more 3D globe visual takes over. Your knowledge of the force can reach a maximum - this is where meta-awareness comes in to know how to best apply intensity to each force for the specific problem at hand. This is to say, critical thinking becomes your meta-awareness.

And once you realise that you are applying your critical thinking skills in the form of meta-awareness of the coding forces, you can measure your own progress and development!

As a Pedagogical Framework

One of the many benefits of this framework, in our very unbiased opinion, is that it is language agnostic and can be adapted with specific examples for whatever language(s) are being taught or used.

As a reflection point, you may consider how code written for teaching and learning purposes is very different from code written for an open shared project or your own projects. This framework provides a model for considering why this is the case and vocabulary for communicating to students, other teaching staff, and even yourself.

We have presented the framework forces in the order in which they are often (unintentionally) taught and experienced by learners. Independent (write or edit code on your own to make it execute without error) > Conscious (understand the bigger picture of your code base and where and how it might be used and make decisions accordingly) > Social (write code for others to read, use, or maintain) reflects the process as a learner and the forces that they may become more aware of as they progress from novice to competent and even intermediate or advanced.

This framework helps students understand why expectations change across tasks. It “allows” exploratory, messy code in learning contexts, while still setting clear standards for shared or assessed work. As an educator, learning how to think about these forces can help you identify where students are struggling and push against forces they may be struggling with. This framework provides a vocabulary to communicate that opens the door for insights and empathy while giving students the same tool.

Rather than teaching a single notion of “good code,” this framework teaches students to employ critical thinking skills and choose how to code based on context. It can be used a means of explicitly communicating to students why the code they find in different places (e.g., GitHub, StackOverflow, etc.) or see in the course may look or feel different and how to make informed decisions on how to model their own code depending on its use case. Shifting from rules to judgment is what shows a transition to more mature, adaptable programmers.

The framework can be used to hang “good coding practices” on, both language specific and agnostic, and highlight **why** they are useful for considered good practice. For example, naming conventions can be good practice aligning with both conscious and social forces. File management allows for independent, conscious, and social forces to all shine. Code comments are an example of how the forces of coding that you are working in can change to make a “good” or “useful” comment. A code comment to yourself is very different from one intended for a colleague to maintain the code base and understand your decision making process.

While we will not go into depth here (to save us all the headache for now), it also provides a language for discussing AI use not as something to accept or reject wholesale, but as a tool whose value depends on whether it supports independence, consciousness, and social responsibility. Using LLMs as a collaborator necessarily means that you must not take what is generated blindly and ensure it is social, which requires conscious and independent coding forces to be in full effect and awareness of the social force. Reading and working with AI generated code also requires some intensity in the social force. Guiding questions if using tools such as LLMs when coding are:

- Do you understand the code that has been generated? Do you understand it well enough to debug it, not just what it is doing?
- Would your colleagues understand this code?
- If the LLM is a collaborator, how do you weigh its suggestions vs other (likely human) collaborator inputs?

Equally, we would suggest that this framework provides a structured answer to ridiculous questions such as *"Why should I learn to code if AI can seemingly generate code for me?"*

Our Answer Would Be Something Along the Lines Of:

That is fine if you want to do that, but you still must use the tool (AI) in a conscious way. You need to know how to prompt, which demonstrates a certain level of independence as you have to know what the ask for in a prompt. This does not demonstrate conscious thought on its own.

In this framework it is your job to change the level of intensity of conscious thought. Similarly, you must intentionally decide what advice to listen to - human collaborator or AI? Should you be "guided" by the AI tool or should you be "directing" the tool? This hits all 3 forces.

But don't take our word for it. Try coming up with your own answer by asking which of the forces the AI is applying. Is it truly applying independent judgement? Is it conscious of the wider needs or implications of your project? Is its code actually as socially aware as it looks at first?

If you would like to hear more about our thoughts in this, get in touch or stay tuned for another chapter in Version 2 of the book.

The Need For This Framework

Yes, none of these forces are a new or unique idea. Many of us are familiar with them already. If you've spent any time around programming education (or lurking on StackOverflow at 2am), you've seen them all before. People talk a lot about things like reading the manual (rtfm)... debugging skills... DRY code (but not too DRY!)... clean code (but not too clean!). And entire books exist for each of these, often with very strong opinions. And yet, many learners struggle to apply them appropriately. The part that's a bit rarer is seeing them all in the same room, talking to each other - or if you will permit, all together on the same seesaw.

More often, they're presented like separate "best practices", each living in its own neat little box. We might have read something about clean code... that leans heavily on social practices, presented as "writing for others", "name things well", or "write clear documentation". But without much mention of the messy, trial-and-error habits that actually allow us to make progress in the first place. Or a debugging guide might encourage us to poke, prod, and experiment freely, but give us little guidance about when it's time to slow down and be more deliberate.

So the advice isn't wrong, but tends to come across as ideas that are isolated from each other, instead of as a set of forces that can be deliberately balanced, where we, ourselves, can decide, rather than guess, how much to lean into a given force at a given time. And that leaves learners to stitch it all together themselves, which is a bit like being given three excellent maps without being told they're of the same place. That's where a lot of the tension creeps in. Beginners, especially, can end up feeling like they're being asked to do contradictory things. "Write clean, self-documenting code" sits right next to "just try stuff and see what breaks," with no explanation of how those ideas fit together. It's not obvious that these

aren't competing rules, but different forces that make sense in different situations.

The Challenge For Educators

Most educators already *feel* this, even if they don't always spell it out. You know that a rough, experimental script written while figuring something out is perfectly fine - even desirable. You also know that the same code would raise eyebrows (or worse) if it showed up in an assessment or shared project. That sense of "what's appropriate here?" develops over time. Learners, however, haven't had time to build those instincts yet. And importantly, they may not even realise there *is* a shift to notice.

A learner who has spent time writing code that is exploratory, maybe slightly chaotic or full of half-formed ideas, comments to self, and creative shortcuts may not understand why those same habits suddenly become a problem elsewhere. It's not that they're being careless, it's just that they're applying the advice they've encountered in the best way they know how... just without a clear sense of how or when to shift their approach as the context changes.

This often shows up in predictable ways. Bits of exploratory code sneak into submissions or shared projects, bringing their weirdness and chaos with them. The result? Code that's harder to read, harder to trust, and harder to build on. On the flip side, some students go the other way by trying to write perfectly polished, fully documented code at all times. Which can be unnecessarily slow and frustrating.

There's also a slightly awkward truth worth saying out loud: **assessment is collaborative.** i.e., the assessor *is* your collaborator. If your code is easy to read, understand, and trust, it's not just "good practice", it will also assess well. For many students, this is their first real experience of writing code for someone else, and without a clear way to frame that shift, assessment criteria can feel a bit arbitrary.

What This Framework Brings

This framework is an attempt to bring some clarity to a muddy reality, not by adding more rules, but by giving people better language.

Instead of treating good coding as a single fixed standard (which we either meet or don't), it frames things as a space we can move around in. Independent, Conscious, and Social aren't boxes to tick; they're forces we can lean into more or less, depending on what we're doing. The more we apply one force to a problem, the more we move our mentality into our desired mode. Deliberately! And conversely, the more we feel ourselves being forced into a corner of a space that isn't working for our problem, the more we can resist that pressure, and move ourselves into a different region of the force-space.

Sometimes we need to experiment quickly and figure things out (Independent). Sometimes we need to slow down and think carefully about structure and decisions (Conscious). Sometimes we need to prioritise clarity for others (Social). Most of the time, we're somewhere in between. And importantly, we can apply conscious thought to independently position ourselves where we need to be.

The aim is to make that movement visible and intentional. Rather than relying on intuition that develops slowly (and unevenly), learners can start to recognise where they are, where they might need to be, and how to adjust.

The radar shows that all three forces are always there, just pulling with different strengths. We don't have to turn one off to use another - we **rebalance** the forces.

This all works regardless of whether we are using R, Python, C, or something more obscure (中文[16] anyone?). The specifics change, but the forces don't, which makes this something we can carry with us across courses, projects, and (eventually) jobs.

16 https://esolangs.org/wiki/%E4%B8%AD%E6%96%87

Hype and The Need for Responsible Compute

Samantha Ahern

Centre for Advanced Research Computing, University College London
s.ahern@ucl.ac.uk

Sam Skipsey

School of Physics and Astronomy, University of Glasgow
Samuel.Skipsey@glasgow.ac.uk

Leo Riviera

Voxa
leo.riviera@voxa.dev

Pawel Orzechowski

Edinburgh Medical School, University of Edinburgh
pawel.orzechowski@ed.ac.uk

Introduction

Hypes come and go, and in their aftermath we are left with rubbish and confusion. We are aware of physical trash in landfills, but rapid technological change also leads to inequality, injustice, and unhelpful social patterns. It is like technical debt (the price we pay later for poor and short-sighted technological decisions now), but on a social level. It is something we can't responsibly ignore as educators and humans.

It's very relevant now that we are in the midst of (another!) AI hype. And if history taught us anything, after summer will come winter, with its sloppy sludge of half-melted mess to deal with. Disturbingly, current narratives of this hype are written by tech bros with a strong (self-serving) agenda. The chorus of this song is 'growth, growth, growth', but growth where and why? We were sold on the dream of machines doing the dishes so that we can write poetry... but instead machines write poetry and we are left with a pile of dirty dishes. Can we afford for art and critical thinking to become on-demand paid subscriptions ('brain as a service') as we outsource our criticality like we delegate our driving to Uber? Is it good for society to replace human companionship with chatbot 'partners'?

> "how do we guarantee that computers and other cultural products are not so pleasurable that they discourage us from engaging in absolutely necessary forms of social engagement?" (Golumbia 2009)

As educators, our responsibility is not to prepare students for the grim post-brain-apocalyptic future, but to inspire them to create a better, sustainable world.

The worst that can happen, is that we educate another generation of self-serving tech bros, who will ruin the world and humanity further. It's irresponsible to teach any skill without also teaching the ethical, social, economical or political context, especially when topics we teach have strong creative and destructive potential (martial arts, nuclear physics, programming). But how do we make these responsibility topics 'not skippable'? In a university setting, ethics can't just be an optional elective subject, or even just a week of content within a core compulsory subject. We need 5 mins of ethics in every hour of teaching, we need a thought about ethics in every sentence. We do not need ethical bits in our yoghurt, we need ethics-flavoured yoghurt.

In this chapter, we look at a number of topics where sustainability, AI, and education (or even wider culture) overlap. We will structure our thoughts around the international framework suggested by the United Nations: the UN Sustainable Development Goals[17]. Join us on this exploration of what is possible, but also of what went terribly wrong in the past. We will look especially at purposefully forgotten (or swept under the carpet) aftermaths of previous technological hypes.

At the end of this chapter, we will bring it back to education and share some practical resources you can use. You will find classroom activities that support responsible computing interspersed throughout, as well as a reading list (see "Summary" below) of our favourite articles and books. Feel free to pick some of them and plant them straight into your own curriculum. And weather-allowing, from these seeds we will watch the growth of a new, curious and slightly more sustainable generation of students and future teachers.

[17] https://sdgs.un.org/goals

Responsible Compute and the UN Sustainable Development Goals

In this section, we will introduce a number of the key issues relating to responsible compute through the lens of the UN Sustainable Development Goals, an outcome of the 2025 UN 2030 Agenda for Sustainable Development[18]. These are a series of 17 goals that address a number of key international issues.

Engineering Inequities



[19]

Failure to consider all members of society in the design and development of products can lead to technologies that privilege some but are not fit for purpose for others. Examples of this include near infra-red automatic soap dispensers, which do not "see" darker skin tones due to their lower reflectivity, and "Shirley" cards in photography, which were designed to help colour balance films but were entirely based on white models (Benjamin 2019). Despite known issues with the film itself and how the printing machines were configured, a diverse "Shirley" card was not introduced until the mid-1990s.

These inequities in film photography have been echoed in facial recognition technologies. For over 50 years, the standard reference image used in image processing papers was "Lena", a cropped image from a November 1972 issue of Playboy. As a result, this image also appears in many course materials and textbooks on image recognition. Use of the image in academic articles has since been banned, due to ethical issues relating to its provenance. In 2015, an issue was reported with the Google Photos app labelling black people as gorillas. Google claimed that this is due to a lack of diverse images of people in its training set, and that the label "gorilla" would no longer be applied to groups of photos while an alternative solution was developed. In 2023, the New York Times reported that the issue was still yet to be resolved, and was also impacting Apple's image recognition (Bandara n.d.).

In 2018, Michael Rosen received a standing ovation at the last ever National Union of Teachers conference after reciting his poem The Data Have Landed[20]. The poem concludes with pupils becoming data, reflecting concerns over the datafication of compulsory education, reiterating sentiments in his previously published short story Alice in Dataland[21]. Social inequities have long been highlighted in popular culture, for example the hip-hop classic White Lines and Bret Easton Ellis' novel American Psycho. In the lyrics to White Lines a business man is treated more favourably than a young black man, despite being arrested for the possessionnof significantly more illegal drugs. In American Psycho, the wealth a power of the main protagonist leads to them literally getting away with murder. Some are repeated in the data driven decision making systems that influence all dimensions of our daily lives, multiple examples of which are highlighted in Virginia Eubanks' 2018 book Automating Inequality (Eubanks 2019).

The free website inequalities.ai[22] provides maps that show how ChatGPT 4o-mini rates regions by comparative properties. The default map shows how it compares countries by "is smarter", placing Europe, North America, China, India and Australasia as "smart", and much of central and western Africa as "not smart" by comparison. This illustrates how bias continues to be an issue with modern, popular "AI" tools.

Classroom Activities: Facial Recognition Workshop

The Programming Historian lesson Facial Recognition in Historical Photographs with Artificial Intelligence in Python[23] introduces key concepts of facial recognition. This tutorial can be expanded to introduce key issues related to inequalities.

- **Example: Blackish, Series 5 Episode 10**. In this episode Diane is "invisible" in her class photo. This can be a starting point to discuss engineered inequity.
- **Example: Lena and Google Photos**. An opportunity to discuss the culture of the research labs where image recognition technologies were developed, how this became part of the "canon", and issues with inadequate training data.

For an example of how these resources can be included, see Ahern (n.d.).

Hardware and the Environment



[24]



[25]

Most environmental discussions relating to data-intensive computing are focused on the direct power costs of running increasingly large and dense collections of compute and storage, with increasing concerns relating to water use in cooling systems.

[18] https://sdgs.un.org/2030agenda
[19] https://sdgs.un.org/goals/goal10
[20] https://michaelrosenblog.blogspot.com/2018/02/the-data-have-landed.html
[21] https://www.theguardian.com/education/2017/dec/19/alice-in-dataland-original-story-education-michael-rosen
[22] https://inequalities.ai/
[23] https://programminghistorian.org/en/lessons/facial-recognition-ai-python
[24] https://sdgs.un.org/goals/goal8
[25] https://sdgs.un.org/goals/goal12

However, the main environmental harms relate to the acquisition of the raw materials required and the embedded carbon in the manufacturing process. **Tantalum, tin, gold**, and other elements are all essential to the production of modern microelectronics. Their minerals are often located in regions with a history of conflict — most notably the Democratic Republic of Congo (DRC) — where demand results in neocolonial forces driving further instability on top of the existing legacies of colonial pasts (Batha n.d.). A rush for gold has led to deforestation in the Amazon and increasing concern about the safety of workers in artisanal mines ("BLOOD GOLD — Is Your Cell Phone or Electric Car Stained with Indigenous Blood from the Amazon?" 2022). Blood gold, which is illegally mined, finds its way into legitimate supply chains with false documentation (Federl, n.d.).

The brine beneath Bolivia's salt flats contain rich deposits of lithium, an essential component in electric batteries. As this is becoming industrialised, there are concerns about who really benefits — the local communities or the companies involved — and about the proposed process and the levels of water required which may impact on neighbouring settlements.

Mining of **coltan**, the dominant source of tantalum (and niobium), is infamous for being deeply exploitative, with hard manual labour the dominant process used in the DRC. As the drive to build data centres has led to natural land being displaced in North America, China, and Europe, so has coltan mining led to deforestation, the near eradication of endangered species (gorillas, in particular, hunted for bushmeat to also feed the miners), and water pollution from dust spoil. There are significant health implications for the workers themselves, with elevated respiratory problems due to that same dust pollution (Leon-Kabamba et al. 2018), and ionising radiation doses approaching the annual exposure limits for UK nuclear industry employees (Mustapha, Mbuzukongira, and Mangala 2007; Agency n.d.).

Classroom Activities: Exploring the Impact of Mining

Current business models are based upon planned obsolescence. This could lead to a discussion of the sustainability of this business model — why the hardware cannot just be manufactured and why computational hardware has a short supported lifespan compared to its actual usable lifespan.

- **Example: River of Gold**.
  - Watch the documentary River of Gold[26]
  - Explore how the Amazon Aid Be a Better Guide[27] relates to the production of electronic hardware
- **Example: Simon Reeve's South America, Episodes 3 & 4**. Episode 3 is in the Andes and features a high altitude gold mine and the working conditions of employees. Episode 4 features the salt flats and lithium extraction.

These issues can also be discussed in workshops relating to code optimisation and green compute. Reasons for optimisation may be for decreasing time to result, or to optimise the amount of social good for the environmental harms.

Data Acquisition and Workforce Exploitation



[28]



[29]

In 2019, the IBM corporation was criticised for using Flickr photos in a facial-recognition project without obtaining the consent of those who had created or were featured in the images (Solon n.d.).

As most images used for machine learning are scraped from the internet and are not well tagged, a large workforce is needed to create this additional metadata. Similarly, tagging of content as illegal or otherwise problematic requires a large amount of initial training data which is manually tagged by humans. To do so at scale, at an affordable cost, AI companies employ outsourcing partners such as Sama, which pay workers in Kenya, Uganda and India a pittance (typically $2/hr) for this work (Perrigo n.d.). This is true even when the content being labelled is known to be obscene and/or deeply distressing — graphic descriptions of sexual assault, for example, or images depicting abuse. To their credit, eventually Sama cancelled some of their worst contracts with AI companies, but other outsourcing companies exist and still perform similar work.

The recent AI boom and the Data Workers Inquiry[30] have highlighted these issues, but concerns regarding Amazon's Mechanical Turk (MTurk) and its workers were being raised back in 2015, and again in 2024 (Dholakia n.d.; Klovig Skelton n.d.). Despite these concerns, MTurk is still used by many researchers across multiple domains.

[26] https://riverofgoldfilm.com/about-the-film/

[27] https://amazonaid.org/be-a-better-consumer/

[28] https://sdgs.un.org/goals/goal3

[29] https://sdgs.un.org/goals/goal8

[30] https://data-workers.org/

Classroom Activities: Data Sources for Training Models

- **Example: Data Workers Inquiry**. Explore "Data Workers in Europe Speak Out"[31]. Do these experiences align with notions of decent work? What are the harms?

- **Example: Hugging Face RAG tutorial**[32]. In this part of the tutorial, you could discuss the source of the file contents for the RAG training. This could then lead into discussions around access and licensing.

Surveillance, Sousveillance, and Civil Liberties



[33]

In 2018, the British Council published an article based on a letter by Tim Berners-Lee (Berners-Lee, n.d.), where he raised concerns about the dark side of the web, alongside his hopes for the future. Specifically, he raised concerns around the concentration of power and the gatekeeping effect which enables a "handful of platforms to control which ideas and opinions are seen and shared".

These fears are now being realised. According to the Fortune 500 Global rankings (Fortune.com 2023), Amazon is the second largest grossing organisation in 2025 and has been in the Fortune 500 Global rankings list for 17 years. Apple is 8th, Alphabet 13th, Microsoft 22nd and Meta Platforms 41st. These companies rely heavily on our data for their platforms and services, producing tools and platforms that shape our everyday lives.

This sentiment is also echoed by contemporary academics. Take David Golumbia, who in the introductory chapter of The Cultural Logic of Computation, asks (Golumbia 2009) "what happens when powerful institutions — corporations, governments, schools — embrace computationalism as a working philosophy". It entrenches power and enmeshes instrumental reason with neoliberalism, despite being framed in terms of distributed power and democratic participation.

In surveillance and sousveillance, the same technologies are used by both those in positions of authority and those wishing to hold them to account. The majority of public spaces are fitted with cameras and other monitoring equipment, many of which have been introduced as public safety initiatives; the few watching the many. There have been concerns related to privacy associated with these technologies — most recently, an audit by the Information Commissioner's Office[34] on the use of facial recognition technology by South Wales Police and Gwent Police. Sousveillance is a form of inverse surveillance, "watching the watchers", facilitated by mobile digital camera technologies and high speed mobile data connections, which has steadily been growing. It can be used either for recording personal experiences, or for the monitoring of authority figures (hierarchical sousveillance) (Reilly n.d.), and can have both positive (e.g. the recording of the George Floyd incident) and negative impacts on individuals (e.g. cyber harassment) and organisations.

Classroom Activities: Responsible AI

- **Example: Datopolis — The Open Data Board Game**[35]. This board game from the Open Data Initiative is available to play online. It introduces key ideas around why data may be open or closed, and longer versions of the game introduce different perspectives on how data can be used. This helps learners to understand that decisions around data use are political and opens up the idea that data itself may not be neutral.

- **Example: Explore whether AI smart glasses should be a consumer technology**. In 2013, Google launched its Google Glass smart glasses. Due to resistance and concerns over use, these were withdrawn from the market in 2015. However, this technology has seen a resurgence. Meta launched AI enabled smart glasses in the autumn of 2025, and Google are considering re-entering the market. However, there are already rising concerns over how these devices are being used.

  A number of campuses in the USA have issued guidance to students over use on campus and threats to safety (Clack n.d.). Smart glasses also pose risks to their wearers: a recent expose revealed that all video recorded on Meta smart glasses is reviewed by data annotators employed by Sama (see "Data Acquisition and Workforce Exploitation"). This is not made clear to users of the glasses, who often record intimate parts of their lives on their product. The Information Commissioner's Office has written to Meta about concerns (Vallance n.d.).

  Students may discuss what the benefit is for Meta and Google in providing smart glasses. Should they be an industrial or a consumer product? This can lead into discussions about corporate responsibility and "surveillance capitalism".

[31] https://data-workers.org/

[32] https://huggingface.co/blog/ngxson/make-your-own-rag

[33] https://sdgs.un.org/goals/goal16

[34] https://ico.org.uk/about-the-ico/media-centre/news-and-blogs/2025/08/ico-steps-up-scrutiny-of-police-use-of-facial-recognition-technology-with-audit-of-south-wales-police-and-gwent-police/

[35] https://datopolis.theodi.org/

- **Example: Explore the AI Incidents Database**[36]. This website is a project of the Responsible AI Collective, an archive of actual harms or near harms of deployed AI systems. The taxonomies of Goals, Methods, and Failures (GMF)[37] and the MIT AI Risk Repository[38] are particularly interesting and could be used when discussing a particular technology or potential for harm.

Cultural Colonialism



[39]



[40]

A related problem to inequity in "AI model training" is inequity in generated content.

Large Language Models generate "plausible" text based on the patterns found in the material they are trained on. However, this material is overwhelmingly English-language, and written by US or British citizens, introducing a cultural bias into the responses from the models themselves (Tao et al. 2023; Jakesch et al. 2023).

As pressure is placed on educational institutions to include "AI" in teaching-related roles, this brings the risk that students learn homogeneous values, potentially those of foreign cultures. This extends not merely to values, but also styles of writing: the "distinctive" tone of an LLM-generated piece of text is a distillation of the dominant styles in its training set (Agarwal, Naaman, and Vashistha 2024). Writers relying on LLMs to "co-develop" essays tend to defer to the LLM's style suggestions relative to their own, even when these stylistic differences are those of culture, not correctness. The effect begins with normative suggestions from spell- and grammar-checkers in text editing software since the 1980s — British English speakers often being "corrected" by default US English software — and also introducing unique cases of language drift when the software encountered words that were not in its dictionary (Zimmer n.d.).

This inhibition of local culture in favour of a "preferred" culture is that of cultural colonisation. This pattern is seen historically in the behaviour of many empires. For example, the British occupation of Ireland during the 16th to early 20th centuries was associated with a suppression of Irish culture and language (Rodriguez and Sojan n.d.), something mirrored in the suppression of Scottish Gaelic and Lowland Scots in Scotland, Welsh in Wales, and the replacement of local languages by English across the British Empire (Lucas n.d.). Cultural homogenisation also occurs within countries (e.g. France (Grégoire, n.d.), Japan (NationCymru n.d.), Turkey, Spain, and many others). Often, these processes are associated with an attempt to impose a dominant culture or politics via language substitution — consider Catalan independence movements in Spain, for example.

With technological promotion, the increased danger is that suppression no longer needs to be forced; instead, the "modernity" and ease of use of the tool acts as a vector to cause adoption. Before LLMs, we saw, and continue to see, "social bubble" effects where online groups iterate themselves into extreme viewpoints ("Social Bubble - ECPS" n.d.); LLMs worsen this by increasingly also homogenising those perspectives in other directions, and by producing content faster than a human can.

Indeed, even before LLMs, the ease with which Internet-based technologies has allowed us to publish content to the entire world has allowed unwitting damage to minority cultures. An American teenager with no Scots knowledge single-handedly degraded much of the Scots language Wikipedia during the late 2010s, with knock-on effects in media representation of the language and culture worldwide (Brooks and Hern n.d.).

Classroom Activities: Generative AI and Homogenisation

- **Example: Volume of content generation**. A Graphite study (Paredes et al. n.d.) shows that LLMs now generate more content on the internet than humans. Students can discuss what this might mean about unique perspectives and the place for human voices and thought. Students may be prompted to consider the danger of conformity in general, and how much we can trust online content without provenance! Educators might discuss the Scots Wikipedia scandal mentioned above as an example of how, even without LLMs, technology amplifies the actions of individuals in potentially harmful ways.

- **Example: Essay "help"**. Students almost certainly have used LLMs to generate or "help" with essay writing or editing. Discuss, with prompting an LLM in class, where individual style may be a matter of taste — and how the style biases of the LLM are themselves obvious. Compare to the diverse range of styles found in (English) literature — do the students think that an LLM would have allowed James Joyce to write Ulysses? Or William Shakespeare to write his plays, given his invention (or at least promotion) of many new words in the English language? (This can become a discussion about the need for critical thinking, creativity and analysis in any text.)

- **Example: Homogenisation beyond text**. Image generation models, such as DALL-E, Stable Diffusion, etc., exhibit biases in the way they respond to prompts (Park 2024). With reference to the images in the linked article, students can discuss the biases they see, or do not see, in the responses; and link them to their own self-image.

[36] https://incidentdatabase.ai/

[37] https://incidentdatabase.ai/taxonomies/gmf/

[38] https://incidentdatabase.ai/taxonomies/mit/

[39] https://sdgs.un.org/goals/goal4

[40] https://sdgs.un.org/goals/goal10

Summary

In this chapter, we have raised a number of repeating socio-technical issues and linked these to the UN Sustainable Development Goals. We have also provided examples of how these topics could be introduced to learners. These examples are directly linked to the QAA Computing Subject Benchmark[41] statement 1.17 regarding sustainability and 1.14 regarding equality, diversity and inclusion, specifically the consideration point: "demonstrating areas in software engineering and application development that require a professional and reflective approach to addressing bias and lack of diversity in data and design of computing solutions, systems and protocols."

We hope that these examples, alongside the QAA and AdvanceHE Education for Sustainable Development Guidance[42], will help embed these key ideas throughout our curricula. In addition, we have provided a list of recommended further reading that will expand on the ideas introduced more fully, and provide inspiration for how we can introduce these key topics to our students and lead to a more sustainable, and ethical, future for all.

Further Reading

- Eubanks (2019), **Automating inequality: how high-tech tools profile, police, and punish the poor.** This is a USA focused book. It discusses a range of issues related to algorithmic decision making that have negatively impacted on less privileged areas of society. The spur for writing the book was Eubanks' partner falling foul of a health insurance algorithm and being denied funding for health care.
- Benjamin (2019), **Race after technology: abolitionist tools for the New Jim Code.** History is littered with technologies that "other"; in this book, Benjamin explores how technologies have the ability to entrench existing inequities whilst appearing at a surface level to be neutral.
- Mejias and Couldry (2024), **Data grab: The new colonialism of big tech and how to fight back.** This book explores data acquisition as a colonial act, comparing it to the land grabs during the age of Empire. It introduces and discusses key topics such as data ownership and individual data rights.
- Jasanoff and Kim (2019), **Dreamscapes of modernity: Sociotechnical imaginaries and the fabrication of power.** Sociotechnical imaginaries are transnational concepts, this was the first text to explore this concept in depth. It introduces the implicit ideas about public purposes, collective futures, and the common good in scientific and technology progress.
- Zuboff (2023), **The Age of Surveillance Capitalism: The Fight for a Human Future at the New Frontier of Power.** This book discusses the business models behind big tech, and how convenience has been traded for privacy. How we have gotten to where we are with data driven technologies and industries in the 21st century.
- Doctorow (2025), **Enshittification: Why Everything Suddenly Got Worse and What to Do About It.** This book more generally discusses the modern phenomenon of "Big Tech" and hypothesises a common pattern to unethical behaviour, whilst suggesting an anti-capitalist response. (Note: this book is available in electronic form, DRM-free, from Doctorow's own web-store[43].)
- Black (2012), **IBM and the Holocaust: The Strategic Alliance Between Nazi Germany and America's Most Powerful Corporation – Expanded Edition.** A meticulously researched piece covering the knowing involvement of IBM (and its German subsiduaries) in the actions of Nazi Germany. A good example of where prioritising profit over ethics once led us, and can lead us again.

References

Agarwal, Dhruv, Mor Naaman, and Aditya Vashistha. 2024. "AI Suggestions Homogenize Writing Toward Western Styles and Diminish Cultural Nuances." In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. Yokohama Japan: ACM. https://doi.org/10.1145/3706598.3713564.

Agency, UK Health Security. n.d. "Ionising Radiation: Dose Comparisons." *GOV.UK*. UK Government. Accessed March 18, 2026. https://www.gov.uk/government/publications/ionising-radiation-dose-

41 https://www.qaa.ac.uk/docs/qaa/sbs/sbs-computing-22.pdf?sfvrsn=ebb3dc81_4

42 https://www.advance-he.ac.uk/knowledge-hub/education-sustainable-development-guidance

43 https://shop.craphound.com

comparisons/ionising-radiation-dose-comparisons.

Ahern, Samantha. n.d. “Image Recognition - History, Ethics and Moral Dilemmas. Workshop Session Plan.”

Bandara, Pesala. n.d. “Google’s Photos App Is Still Unable to Find Gorillas.” *PetaPixel*. Accessed March 18, 2026. https://petapixel.com/2023/05/22/googles-photos-app-is-still-unable-to-find-gorillas/.

Batha, Emma. n.d. “How Murky Gold Trade Fuels Wars, Gangs and Terror Attacks Context by TRF.” Accessed March 18, 2026. https://www.context.news/money-power-people/how-murky-gold-trade-fuels-wars-gangs-and-terror-attacks.

Benjamin, Ruha. 2019. *Race After Technology: Abolitionist Tools for the New Jim Code*. Cambridge, UK Medford, MA: Polity.

Berners-Lee, Tim. n.d. “’The Web Is Under Threat’: Tim Berners-Lee on Making the Web Work for Everyone.” *British Council*. British Council Worldwide. https://www.britishcouncil.org/anyone-anywhere/explore/dark-side-web/tim-berners-lee.

Black, Edwin. 2012. *IBM and the Holocaust: The Strategic Alliance Between Nazi Germany and America’s Most Powerful Corporation – Expanded Edition*. Dialog press.

“BLOOD GOLD — Is Your Cell Phone or Electric Car Stained with Indigenous Blood from the Amazon?” 2022. *The People’s Forum*. https://peoplesforum.org/events/blood-gold/.

Brooks, Libby, and Alex Hern. n.d. “Shock an Aw: US Teenager Wrote Huge Slice of Scots Wikipedia.” *The Guardian*. Accessed March 18, 2026. https://www.theguardian.com/uk-news/2020/aug/26/shock-an-aw-us-teenager-wrote-huge-slice-of-scots-wikipedia.

Clack, Erin. n.d. “University Issues Warning After Reports of a Man Using Ray-Ban Meta Sunglasses to Record Women on Campus.” *People.com*. Accessed March 18, 2026. https://people.com/university-issues-warning-after-reports-of-man-using-ray-ban-meta-sunglasses-to-record-women-on-campus-11824196.

Dholakia, Utpal. n.d. “My Experience as an Amazon Mechanical Turk (MTurk) Worker Psychology Today United Kingdom.” Accessed March 18, 2026. https://www.psychologytoday.com/gb/blog/the-science-behind-behavior/201507/my-experience-as-an-amazon-mechanical-turk-mturk-worker.

Doctorow, Cory. 2025. *Enshittification: Why Everything Suddenly Got Worse and What to Do about It*. Verso Books.

Eubanks, Virginia. 2019. *Automating Inequality: How High-Tech Tools Profile, Police, and Punish the Poor*. First Picador edition. New York: Picador St. Martin’s Press.

Federl, Fabian. n.d. “Investigating How Illegal Gold Gets Into the Legitimate Supply Chain.” *Global Investigative Journalism Network*. https://gijn.org/stories/investigating-illegal-gold-mining-legitimate-supply-chain/.

Fortune.com. 2023. “Fortune Global 500.” *Fortune*. https://fortune.com/ranking/global500/2023/.

Golumbia, David. 2009. *The Cultural Logic of Computation*. Cambridge, Mass: Harvard University Press. https://doi.org/10.5860/choice.47-2615.

Grégoire, Henri. n.d. “Rapport sur la nécessité et les moyens d’anéantir les patois et d’universaliser l’usage de la langue française — Convention nationale, 1794.” *Wikisource*. https://w.wiki/KrJV.

Jakesch, Maurice, Advait Bhat, Daniel Buschek, Lior Zalmanson, and Mor Naaman. 2023. “Co-Writing with Opinionated Language Models Affects Users’ Views.” In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. Hamburg Germany: ACM. https://doi.org/10.1145/3544548.3581196.

Jasanoff, Sheila, and Sang-Hyun Kim. 2019. *Dreamscapes of Modernity: Sociotechnical Imaginaries and the Fabrication of Power*. University of Chicago Press.

Klovig Skelton, Sebastian. n.d. “Amazon Mechanical Turk Workers Suspended Without Explanation Computer Weekly.” News. *ComputerWeekly.com*. Accessed March 18, 2026. https://www.computerweekly.com/news/366612812/Amazon-Mechanical-Turk-workers-suspended-without-explanation.

Leon-Kabamba, Ngombe, Nlandu Roger Ngatu, Sakatolo Jean-Baptiste Kakoma, Christophe Nyembo, Etongola Papy Mbelambela, Reiko Juliana Moribe, Stanislas Wembonyama, Brigitta Danuser, and Numbi Oscar-Luboya. 2018. “Respiratory Health of Dust-Exposed Congolese Coltan Miners.” *International Archives of Occupational and Environmental Health* 91 (7): 859–64. https://doi.org/10.1007/s00420-018-1329-0.

Lucas. n.d. “How Colonialism Spread the English Language.” *English Project*. Accessed March 18, 2026. https://englishproject.org/how-colonialism-spread-the-english-language/.

Mejias, Ulises A, and Nick Couldry. 2024. “Data Grab: The New Colonialism of Big Tech and How to Fight Back.” In *Data Grab*. University of Chicago Press.

Mustapha, A O, P Mbuzukongira, and M J Mangala. 2007. “Occupational Radiation Exposures of Artisans Mining Columbite–Tantalite in the Eastern Democratic Republic of Congo.” *Journal of Radiological Protection* 27 (2): 187–95. https://doi.org/10.1088/0952-4746/27/2/005.

NationCymru. n.d. “Japanese Language Subjected to ’Welsh Not’-Style Punishment Takes Inspiration from Cymraeg.” *Nation.Cymru*. Accessed March 18, 2026. https://nation.cymru/culture/japanese-language-subjected-to-welsh-not-style-punishment-takes-inspiration-from-cymraeg/.

Paredes, Jose L., Ethan Smith, Gregory Druck, and Bevin Benson. n.d. “More Articles Are Now Created by AI Than Humans.” Accessed March 18, 2026. https://graphite.io/five-percent/more-articles-are-now-created-by-ai-than-humans.

Park, Ye Sul. 2024. “White Default: Examining Racialized Biases Behind AI-Generated Images.” *Art Education* 77 (4): 36–45. https://doi.org/10.1080/00043125.2024.2330340.

Perrigo, Billy. n.d. “Exclusive: The $2 Per Hour Workers Who Made ChatGPT Safer.” *TIME*. Accessed March 18, 2026. https://time.com/6247678/openai-chatgpt-kenya-workers/.

Reilly, Paul. n.d. “Sousveillance.” In *The Routledge Encyclopedia of Citizen Media*, 1st ed. London: Routledge. Accessed March 18, 2026. https://www.taylorfrancis.com/books/9781315619811/chapters/10.4324/9781315619811-65.

Rodriguez, Pamela, and Kiran Sojan. n.d. “How the English Erased Irish Kairos Magazine.” Accessed March 18, 2026. https://kairosmagazine.rutgers.edu/killing-off-irish/.

“Social Bubble - ECPS.” n.d. Accessed March 18, 2026. https://www.populismstudies.org/Vocabulary/social-bubble/.

Solon, Olivia. n.d. “Facial Recognition’s ’Dirty Little Secret’: Social Media Photos Used Without Consent.” *NBC News*. Accessed March 25, 2026. https://www.nbcnews.com/tech/internet/facial-recognition-s-dirty-little-secret-millions-online-photos-scraped-n981921.

Tao, Yan, Olga Viberg, Ryan S Baker, and René F Kizilcec. 2023. “Cultural Bias and Cultural Alignment of Large Language Models.” *PNAS Nexus* 3 (9). https://doi.org/10.1093/pnasnexus/pgae346.

Vallance, Chris. n.d. “ICO Writes to Meta over ’Concerning’ AI Smart Glasses Report.” *BBC News*. Accessed March 18, 2026. https://www.bbc.co.uk/news/articles/c0q33nvj0qpo.

Zimmer, Ben. n.d. “When Spellcheckers Attack: Perils of the Cupertino Effect.” *OUPblog*. Accessed March 18, 2026. https://blog.oup.com/2007/11/spellchecker/.

Zuboff, Shoshana. 2023. “The Age of Surveillance Capitalism.” In *Social Theory Re-Wired*, 203–13. Routledge.

An Optimistic Outlook on Teaching, Learning and Assessment for Coding With the Emergence of Generative AI

Steven Watterson

Personalised Medicine Centre, School of Medicine, Ulster University
s.watterson@ulster.ac.uk

Laila Dabab Nahas

Edinburgh Medical School, University of Edinburgh
ldababn@ed.ac.uk

Tuğrulcan Elmas

School of Informatics, University of Edinburgh
tugrulcan@ed.ac.uk

Hebatallah Shoukry

School of Engineering and Physical Sciences, Heriot-Watt University
h.shoukry@hw.ac.uk

Ozan Evkaya

School of Mathematics, University of Edinburgh
Ozan.Evkaya@ed.ac.uk

Introduction

The introduction of generative AI (GenAI) has transformed how staff and students view teaching, learning and assessment. Whilst education is a diverse sector, there has been universal recognition that GenAI has the potential to derail teaching and learning activities (Bente, Randall, and Wäckerle 2024; Brennan and McDermott 2023). The traditional paradigm of communicating ideas, tutoring understanding and assessing learning has become harder to safeguard as authentic when GenAI tools can synthesize text rapidly and easily to a level consistent with passable understanding. Were GenAI tools highly exclusive with limited availability then that risk might be modest, but they have been made widely and freely available, by technology companies that students already trust, as those companies look to build their user bases.

In light of the impact that GenAI will have on teaching, learning and assessment and recognising that it is likely to become part of the graduate's tool set, it is worth considering how we can adapt teaching both to support learners to develop their skills with GenAI tools, but also to defend the authenticity of learning and assessment as far as possible. Here, we consider strategies that can be followed to incorporate GenAI into teaching and to adapt assessment to mitigate the risk from GenAI-based fraudulent completion. To understand the rationale for these strategies, it is important to have some understanding of how GenAI tools work.

How Does Generative AI Work?

GenAI takes a mathematical approach to describing language. Sentences can be broken up into words, partial words, and elements of punctuation. These components are known as tokens and the combination of tokens that make up a line of text is mapped to a data point on a set of axes (in a high number of dimensions). Some processing is applied when creating the data point that weights the relative importance and order of the tokens in the text. Similar text will be mapped to nearby data points, whereas very different text will be mapped to data points that lie far apart (Hua and Yao 2024). The words, partial words, and punctuation that the GenAI recognises as tokens are its vocabulary and the data point describing the text is fed into a neural network. Then calculates the probabilities of each token next following the text, where the neural network itself has been trained with large volumes of literature[44] (Jiang et al. 2019). The GenAI algorithm doesn't always pick the highest probability next token, but one of the most probable and how much it is allowed to deviate from the most probable choice is a model parameter that ensures some variability in the responses generated. Once the next token has been selected, it is added to the text, a new data point is calculated and this is fed into the neural network, with the process repeating until a suitable volume of text has been generated or a stop signal is reached (Ponmalar et al. 2025).

It is worth emphasizing that the text generated is a statistical construction based on the training literature, not a series of quotes. This means that factually correct sentence structures cannot be guaranteed and are only likely to be generated when they are common in the training literature. Factually correct sentences about

[44] https://hatchworks.com/blog/gen-ai/large-language-models-guide/

uncommon topics are harder to produce consistently (Jiang et al. 2019).

This system is known as a Large Language Model (LLM) and it forms the basis of most GenAIs. One quirk of LLMs is that they are trained to consider the input text, which is formed from the prompt, as factual and to reference it with little statistical reinterpretation. Furthermore, prompts can be extremely lengthy (Gan and Mori 2023). We can combine these two properties of LLMs to improve the factual accuracy of the generated text by including factual source materials as part of the prompt. This is usually done in a process called Retrieval Augmented Generation (RAG) and it is a way to dramatically improve the factual recall of the generated text. By using RAG, the LLM can be formulated so that it generates narrative text fluently based on its training data, but draws on factual details from source materials through RAG (see Figure 9.1). It yields a system that can be both conversational and factually accurate (Miao et al. 2024).

The Current State of GenAI in Learning and Teaching

We are still in the early stages of exploring the implications for GenAI in teaching and learning. The literature reveals contradictions in how GenAI tools are employed in education. From one perspective, GenAI offers promising benefits when integrated into educational settings under the guidance of educators using factual frameworks such as RAG. Some systematic studies have shown that GenAI can facilitate student learning, enhancing cognitive, and technical and interpersonal skills, and students who used GenAI performed better than those who did not (Daniel, Msambwa, and Wen 2025; Heung and Chiu 2025). Additionally, students using GenAI outperform those who do not in areas such as learning performance, AI awareness, and managing cognitive load. However, students who do not use GenAI tools demonstrate better critical thinking (Ji et al. 2025). Whilst critical thinking is ordinarily superior amongst non-GenAI-using student groups (Ji et al. 2025), when GenAI is used appropriately improvements in critical thinking have been shown (Daniel, Msambwa, and Wen 2025). It has been recommended as best practice to encourage students to first attempt tasks independently before consulting GenAI tools (Kosmyna et al. 2025).

From the other perspective, questions remain about how to deal with the unguided use of these tools, which can lead to over-reliance and missed learning opportunities for students who over-delegate to GenAI tools (Kosmyna et al. 2025; Pearson 2025). This phenomenon, known as “cognitive offloading”, occurs when individuals choose not to commit details to memory or invest effort in learning skills. Instead, they relying heavily on GenAI tools to retrieve data and complete tasks for them (Pearson 2025; Fan et al. 2024). These challenges are not new. With the advent of search engines, it was documented (Pearson 2025) that there was a tendency to memorise less information, instead counting on search engines to facilitate the retrieval of web-based data as a means of recall. As GenAI grows in quality and capability, there is a risk that cognitive offloading may go beyond recall and result in entrusting entire tasks to AI irresponsibly (Fan et al. 2024). Hallucinations and the biases gained in training GenAI tools both present the risk of unintentionally adopting incorrect information and there is concern over the influence of the companies providing the GenAI tools, where accidental or deliberate training biases could affect public usage (Pearson 2025).

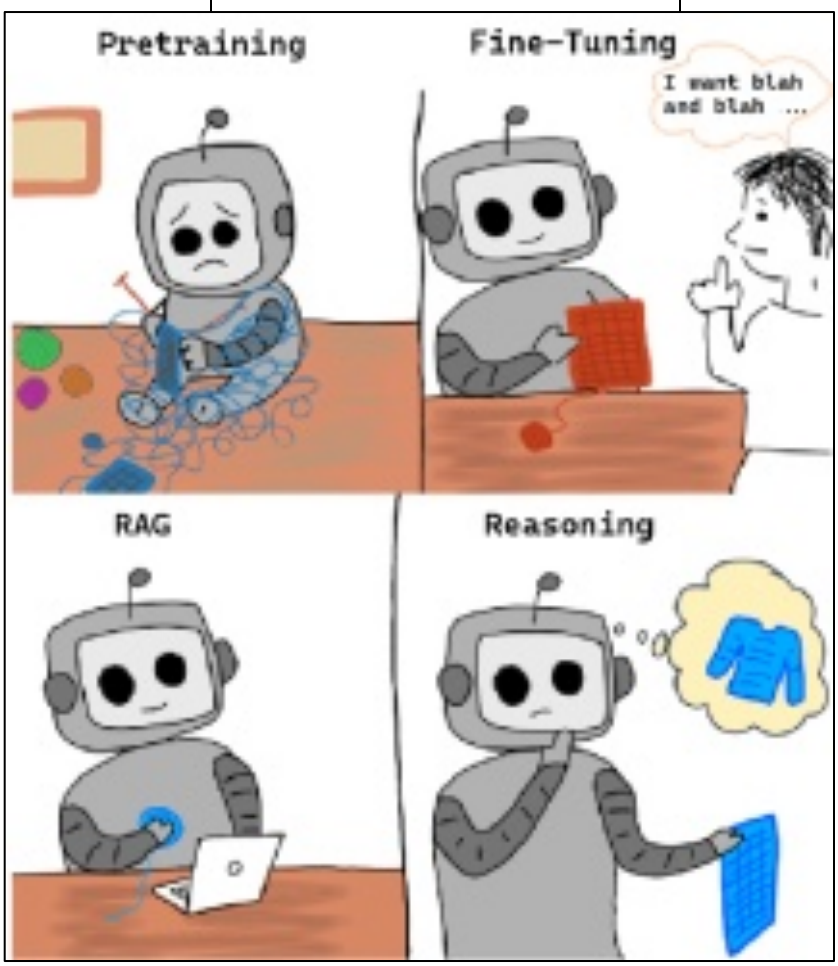


Figure 9.1:

Top:

LLMs development through a two-step process: pretraining & finetuning with question answering in finetuning where the user asking GenAI to do something (e.g., in this example a metaphor for a user asking the GenAI to create a knitted orange square from wool and GenAI did it).

Bottom: enhancement of LLMs through RAG & Reasoning.

The key differentiator in outcomes is how GenAI is employed. Without proper guidance students can over-rely on GenAI tools, substituting the generated output for their own ideas. It is essential to equip students with strategies to harness GenAI's potential whilst avoiding its pitfalls (Boers et al. 2025). Many studies have emphasized the need for longitudinal research to understand the long-term effects of GenAI on recall and cognitive skills, but in the absence of longitudinal research, we must continue to innovate, develop and adapt the guidance and best practices that will support students effectively in integrating GenAI into their academic and professional lives.

GenAI as a Teaching and Learning Partner

GenAI has enormous potential to support students as a teaching and learning partner, offering detailed explanations, engaging in discussions, and interactively responding to queries 24 hours a day, 7 days a week. This makes it a valuable tool that can fulfill a wide range of roles.

GenAI As Topic Expert

By using Retrieval-Augmented Generation (RAG) to integrate a factual corpus of knowledge into GenAI responses, it becomes possible to consider employing GenAI tools as expert systems that can support both staff to teach and students to learn. As a teaching assistant, GenAI has the potential to support staff to create more learning materials across more learning modalities than is currently possible in limited time, which can enhance support for students with different learning preferences (Raju 2025). GenAI has the further potential to act as the first receiver for student questions and queries. By increasing the productivity of teaching staff and by resolving the more straightforward queries, GenAI tools have the potential to help teaching staff to increase the time committed to key pedagogical task (Nyaaba et al. 2024).

The proximity of data points in vector space can represent related ideas. Due to that, GenAIs can connect related ideas well and are adept at providing alternative explanations (as a result of their inherited randomness). GenAIs, therefore, can perform well in tutoring roles and have been well received in studies of student adoption (Kanont et al. 2024; Bouzid and Piron 2024; Watterson et al. 2025). GenAI also has the potential to directly benefit the student by acting as a topic expert available 24/7 that can interact with the student and expand on the areas of greatest need, providing truly personalised support. With access to course documentation, it can also help with management of deadlines and timetables, etc. Examples 1 and 2 below demonstrate use of Google Gemini 2.5 Flash and NotebookLM on student queries, with additional information provided by RAG.

Example 1: Administrative Information Provided by RAG to a LLM

Input:

Module Handbook

- Degree Programme: Programming exercises for coders.
- Module Code: 1234.
- Office Hours: Tuesday and Thursday 2pm-4pm.
- Module Coordinator: Hamish McBeth.
- Room Number: 12.
- Course Duration: 12 weeks.
- Assessments: Examinations are in weeks 6 and 12. Coursework deadlines are at the end of week 5 and the end of week 11.

Prompt: Who is the coordinator and where and when can I find them?

- Response: The Module Coordinator is Hamish McBeth. You can find him in Room Number 12 during his office hours, which are Tuesday and Thursday from 2pm to 4pm.

Example 2: Code Provided by RAG to a LLM

Input:

```
import numpy as np, pandas as pd
from sklearn.datasets import
load_iris
from sklearn.decomposition import
PCA

iris = load_iris()
pca = PCA(n_components=2)
principalComponents =
pca.fit_transform(iris.data)

print(f"Explained variance by PC1
is ",
pca.explained_variance_ratio_[0],
f"Explained variance by PC2 is ",
pca.explained_variance_ratio_[1])
```

Prompt: Concisely, what does this code do and what does it print out?

- Response: The code performs Principal Component Analysis (PCA) on the Iris dataset, reducing its dimensionality to two principal components. It then prints the explained variance ratio for PC1 and the explained variance ratio for PC2.

AI as a Coding Partner

An effective paradigm used to teach coding is that of paired programming where two coders collaborate on a single script. Each takes a separate role, with one acting as the primary code-writer (driver) and the other acting to spot errors, provide structural feedback and manage the resources needed (navigator). By separating the coding activity into writing and reflective roles, paired programming can improve the efficiency of code delivery; by alternating roles, both partners have the opportunity to learn as they work.

Where a coder does not have a coding partner, a GenAI can take on the role. As a writing partner, GenAI tools can create code for a precise prompt. GenAI tools are more accurate and less prone to syntactical errors than humans (i.e., they are less likely to forget a comma or bracket or miss a function specific parameters), but they can lack sufficient context to ensure that the code meets every explicit and implicit requirement of a brief or that the blocks of code generated independently can integrate adequately. These errors can be addressed by the coder. As a reflective partner, GenAI tools can evaluate code, recognise structural errors and suggest refinements to a specific remit to support the coder; see Example 3 below. In both cases, the quality of the role-play will be very sensitive to the prompt and so some trial and error may be required. It may be advantageous to separately specify roles in system-level prompts, which are commonly used to frame the subsequent dialogues.

Example 3: Prompted Critique for the code in Example 2 (Gemini 2.5 Flash/NotebookLM)

Prompt: Please concisely critique this code.

Response: This code snippet from "Iris Dataset PCA Explained Variance" effectively demonstrates the application of Principal Component Analysis (PCA) to the Iris dataset. Here's a concise critique:

Positive Aspects:

- Clarity and Purpose: The code is straightforward, clearly showing how to load data, initialise PCA with two components, and then fit and transform the data to calculate the explained variance ratios.
- Standard Libraries: It uses widely accepted libraries such as sklearn.datasets, sklearn.decomposition, and numpy for data loading, PCA, and numerical operations, respectively.
- Direct Output: The print statement effectively displays the explained variance for the first two principal components (PC1 and PC2).

Areas for Improvement:

- Unused Imports and Variables: The pandas library is imported (import pandas as pd) but not utilised in the provided code snippet. Similarly, the `principalComponents` variable, which stores the transformed data, is created but not used for any subsequent operations within this specific excerpt.
- Scope: While effective for its stated purpose, the snippet could be extended with visualisations (e.g., a scree plot) or further analysis to demonstrate the practical application of the principal components for dimensionality reduction.

A number of Integrated Development Environments (IDEs) already include Ais as a coding partner and this approach is regularly used in industry. Developer surveys and interviews have recorded improvements in code quality and developer satisfaction, but with questions over code reliability, explainability, and trust (Bughin 2024; Chen 2024). Some studies have shown that paired programming with a GenAI tool delivers the best assignment outcomes for students compared to human paired programming or solo programming (Bughin 2024; Lyu et al. 2025). However, other studies have suggested that the improvements in the speed with which code is created are offset by the additional time needed to debug the code (Imai 2022), and that concerns about sensitive parts of code bases mean that GenAI tools cannot support projects consistently, only in a piecemeal manner (Coutinho et al. 2024).

AI as a Puzzle Setter

GenAIs can be prompted to generate coding questions along with worked solutions. This can be invaluable as a way to efficiently produce formative exercises and it becomes very powerful when pre-existing scripts or code-bases are included by RAG to use as a reference document (Pourcel et al. 2024). Typical exercises can include (i) generating incomplete blocks of code for students to complete, (ii) Parsons puzzles where students rearrange instructions into the correct sequence, or (iii) asking students to interpret the behaviour of a generated block of code (see Example 4) (Pourcel et al. 2024; Hou et al. 2024). This approach allows formative coding exercises to be generated with relative ease that align directly to topics of study, such as a particular algorithm or data set.

Example 4: Using the GenAI Tools To Create Coding Puzzles From the Script Provided by RAG in Example 2 (Gemini 2.5 Flash/NotebookLM)

Prompt: Produce a version of this code with key words missing that can be used as a teaching and learning exercise.

Response:

```
import numpy as np, ____ as pd}
from sklearn.datasets import ____
from sklearn.decomposition import
____

iris = load_iris()
pca = PCA(n_components=____)
principalComponents =
pca.____(iris.data)

print(f"Explained variance by PC1
is ", pca.____, f"Explained
variance by PC2 is ", pca.____)
```

Prompt: Please rearrange this code in a parsons puzzle.

Response: Parsons Puzzle: Reconstruct the PCA Code — Please arrange these lines of code in the correct sequence to perform PCA on the Iris dataset and print the explained variance ratio. Pay attention to the order and any necessary indentation.

```
1. from sklearn.datasets import
load_iris
2. pca = PCA(n_components=2)
3. print(f"Explained variance by
PC1 is ",
pca.explained_variance_ratio_,
f"Explained variance by PC2 is ",
pca.explained_variance_ratio_[1])
4. principalComponents =
pca.fit_transform(iris.data)
5. import numpy as np, pandas as
pd
6. iris = load_iris()
7. from sklearn.decomposition
import PCA
```

How to Solve: Your task is to place these lines in the logical order of execution for the script. The original script begins with imports, then loads data, initializes the PCA model, performs the transformation, and finally prints the results

GenAI as a Coding Tool for the Student

As the use of GenAI tools to create code grows, the role of the coder will necessarily change. The coder will no longer directly produce code, but instead supervise the tools generating the code. Nonetheless, the requirement to stand over the code and take responsibility for its correctness and accuracy will remain. This can be achieved either (i) by forensically working through the code produced by the GenAI tools or (ii) modularising the challenge, having the GenAI tools separately produce code for each module and then evaluating the performance of each module's code separately and in combination. The latter is a form of unit testing; a process widely used in industry to manage complex projects across coding teams. Here, it transforms the student into a code manager (Trautsch and Grabowski 2017; Osherove 2009). This way of working is both likely to scale better as projects become increasingly sophisticated, but also has the benefit of starting students to think and work in ways that are closer to software roles in industry (Vorel 2025; Bhatia et al. 2023). In addition, it is very important to notice that a code manager is not for first or second years students but for more senior students after fully understanding and acquiring the foundational skills to spot mistakes.

A unit testing approach means introducing a formal design stage where projects are broken into constituent modules/units with clear specifications (Osherove 2009; Northwood 2018) in which written modules/units are (i) configured with a state and data on which to run (Arrange), (ii) run on the data and state (Act) and (iii) evaluated on the correctness of the output (Assert). A Test Runner is then constructed to evaluate a range of inputs and outputs for each module.

A basic unit testing framework evaluates the behaviour of the code, not the code itself, which is well suited to contexts where coding can be entirely devolved to GenAI tools. Where the use of particular commands or data structures is a requirement, the tests must be adapted to reflect this. Ensuring that the tests are rigorous and appropriate will become a new skill and responsibility for the student.

Unit testing is not only a tool for students to use in code development, but also a tool for staff to use when assessing student submissions and significant work has been undertaken evaluating how well GenAI tools can create unit tests from project briefs (Bhatia et al. 2023; Mock, Melegati, and Russo 2024). There, GenAI such as ChatGPT, Copilot and Gemini were studied in a variety of programming languages such as Python and Java. It was found that such tools perform well, especially in terms of repetitive tests. However, the literature stresses the necessity of human oversight for correctness because of its weaknesses, such as incorrect or hallucinated assertions and weak reasoning about program logic so it requires significant human review.

Assessing Learning

Assessment should always evaluate whether learning has taken place, but, depending on the format, there is significant potential for GenAI to be used fraudulently. The ease of use and high availability of GenAI tools means that conventional take-home, open book, step-wise assignments are highly vulnerable to completion by GenAI tools.

Whilst it is impossible to devise assessments that are invulnerable to GenAI tools, especially when they must remain accessible to students of a wide range of levels of academic ability, there are strategies we can follow that increase the difficulty and therefore reduce the likelihood of students using GenAI in their submissions. Across the range of academic abilities, the most capable students will feel that they have

the least to gain and the most to lose from fraudulently using GenAI. At the other extreme, the least academically able students will feel that they have the most to gain and the least to lose. However, the least able students are also often those who are the least invested in their learning. Our guiding principle to making assessments resistant to GenAIs, is that we must design assignments so that the workload required to complete it by GenAI, ideally exceeds the workload of making an honest submission, but at least exceeds the level of work that less capable students are prepared to commit. Nonetheless, a motivated and capable student will always be able to engineer prompts that support the completion of an assignment. Thus, each of those techniques comes with challenges such as the better start-bias, where students start learning at different levels which we need to take into account and appreciate when developing these strategies. We will consider several strategies here and for a particular exercise, a combination of strategies may be optimal.

Supervision of Assessment

Amongst the simplest ways to maintain the authenticity of assessment is to enforce strict supervision. This is a time- and labour-consuming approach and is not necessarily an efficient use of resources, especially for longer and more complex assignments.

A more time-efficient version of this approach involves replacing complex assignments that evaluate the coordinated use of multiple learning outcomes with a series of shorter competency-based assessments each testing a single atomised component of the learning outcomes. This might be a series of tests in which a student is asked to demonstrate core elemental coding skills whilst being observed one-to-one by the assessor. Each assessment for each student may require only a short time, with a binary pass/fail outcome, and progression from the programme then becomes dependent on a portfolio of demonstrated competencies.

Insufficiency

We can frustrate GenAI by designing assignments with briefs that are individually incomplete and require knowledge from multiple sources to be made complete. Such an assessment must be designed so that a student must bring understanding gained elsewhere to the assessment to fully understand the brief. For a student to use GenAI tools successfully on the assessment, they must both piece together the missing information from the other sources and also formulate it in a way that complements the content of the brief and can be interpreted unambiguously by the GenAI tool. Whilst this will be attainable for the most capable students, returning to the principle described above, we only need to design an assessment so that compiling the brief and the missing information into successful prompts incurs a greater workload than a less capable student is prepared to commit.

Designing assignments that bring together learning from multiple sources, potentially across different modalities (e.g., using videos, written text and external resources), maximises the workload in completing the assignment brief and therefore the challenge of building a useful prompt for a GenAI tool. In practice, incompleteness can take the form of ambiguous phrasing, ambiguous technical detail, missing stages of a pipeline, or newly introduced terminology with resolutions and definitions embedded elsewhere in the programme (for example, having the general in text while more details are in another form such as a video). What inhibits a student from using GenAI tools successfully in this case is that they must not only piece together the missing information needed to complete the assessment brief, but also formulate it in a way that is useful and unambiguous for GenAI completion.

Finally, teachers must be aware that some risks could arise from such approaches for developing such assessments. For example, giving extra attention to material accessibility for some students with special needs and trying to avoid extra unnecessary pressure could be added to students as barriers, such as extra time or effort to put in order to complete the assessment.

Misdirection

We can construct assignments so that they are interpreted differently by students and by GenAI. The simplest example is where hidden text is placed in an assessment brief. Typically, this is text embedded in a paragraph at a small and unobtrusive font size, coloured to blend into the background. This hidden text will not be seen by the student who manually attempts the assessment brief, but when the brief is copied or imported into a GenAI tool, the hidden text is exposed and treated as a part of the brief. The hidden text can negate a statement, add digits to critical numbers or misdirect the GenAI and whilst it will be observed by a diligent student, it may go unobserved when less care is taken, especially in longer briefs where the hidden text can become harder to discern.

Critically, the hidden text should be chosen so as to lead to incorrect responses when the brief is imported or copied into a GenAI prompt without careful attention. This may help in making it more complicated for students who are using GenAI to obtain sufficient marks without extra care and true engagement with the assessment. We can not consider this an effective way for detecting GenAI misuse but we can make it harder for those who misuse GenAI to get undeserved accomplishments.

Reverse Engineering Assessments

The principal strength of GenAI tools lies in their ability to follow a sequence of steps and instructions. Assessments that move away from asking students to complete a prescribed sequence of steps and instead to explore permutations of options to identify an optimal solution are therefore likely to be inherently more challenging for GenAI tools to complete.

One such approach is to ask the student to reverse engineer an analysis or technique. Providing a student with a starting data set, the output from an analytical pipeline and an understanding of the possible steps

that could be combined to derive the output from the starting data set. It is possible to design assessments in which the student must experiment to identify the combination of steps that generate the output from the starting data. If this pipeline has been completely explained elsewhere, this becomes a specialised case of an insufficiency assessment where part of the instructions can be retrieved from elsewhere in the programme. However, if the sequence of steps have not been taught together, but only individually and with alternatives having been explored, the student will have to evaluate multiple permutations of options. Depending on the explicitness of the assessment brief, the number of steps that must be combined and the number of options available at each step, GenAI tools are unlikely to enumerate all possible options to identify the correct answer.

Unsighted Data Sets and Structures

GenAIs are exposed to large volumes of pre-existing scientific literature as part of their training. Assessments that are designed based on widely published data sets or data structures are likely to be easier for GenAI tools to complete than assessments based on rare or bespoke data sets and structures. GenAIs are sensitive to prompts and will attempt to create code for data sets and structures described in the prompt, but if the assessment data sets and structures are sufficiently uncommon then the differences between this and the sets and structures in the related training data may cause the GenAI tools to code imperfectly, with errors for the rare and bespoke examples. Students will then be challenged and need to demonstrate competency to debug these errors. An effective, rare source of data might be that held within databases as this is less likely to have been scraped as part of a training set.

This approach comes with the caveat that experience of working with standardised data sets and structures are often an important learning outcome and so directing students towards the rare and bespoke and away from standard data sets and structures may not be constructive or translational.

Student Self-Assessment

Whilst it is strategically important to frustrate the use of GenAI in assessment, at the same time, we must acknowledge that GenAI tools have a place supporting student learning and we need to encourage students to use AI constructively to prepare them for the workplace. When used effectively, AI can enhance productivity, ensure continuous skill development and provide a more secure future aligned with emerging technologies.

Asking students to outline the logical steps they planned and followed using AI tools as an assistant for completing tasks or assessments introduces a certain degree of transparency around GenAI engagement (Cotelli Kureth, Paliot, and Zink 2025). They should describe the steps taken to validate GenAI outputs, as well as how they addressed biases and disparities in the results. A self-assessment provides students with the additional opportunity to reflect on their experience of working with GenAI tools, identifying when GenAI use was most beneficial, when it was not or was unproductive and how it has improved their learning skills and helped them to achieve their goals (Combrinck and Loubser 2025). Checklists have already been introduced to students to disclose and reflect on their use of GenAI either as part of their assessment or as general guidance. We recommend the example provided by the University of Leeds and published in their generative AI guidance pages[45].

The Digital Divergence

The introduction of any new technology can create inequalities between those who are skilled and able to exploit the technology and those who aren't. However, conventionally, new technologies have been peripheral to the main tasks of coding and so have only offered incremental benefits to productivity. The great potential for GenAI tools lies in their ability to directly create code and therefore accelerate productivity. This is qualitatively different to productivity boosts seen before (e.g., before it used to be manually catching typos and errors, while now it could be easily highlighted and caught with different editors or tools supported by GenAI).

Skilled use of GenAI tools will not only affect the pace at which code is created, but also the scale, so that code can be created quickly and more easily in parallel. These developments will necessarily place an emphasis on code management and prompt engineering, which will have to be taught alongside the fundamentals of coding. These new paradigms raise the level at which a student will be expected to operate and create new standards of excellence for student learners. However, adding new and more layered skills to a teaching programme is likely to exacerbate the academic divide between weaker and stronger learners (Suárez and García-Mariñoso 2025; Rottner et al. 2025).

At the same time, because GenAIs have weakened the legitimacy of the simplest structures of assessment, not only is the syllabus likely to develop in sophistication, but so too will the structures and formats of assessment, potentially compounding confusion and further exacerbating the academic gap between weaker and stronger learners.

[45] https://generative-ai.leeds.ac.uk/ai-and-assessments/gen-ai-quick-checklist/

Conclusion

Here we have introduced GenAI as both an asset to teaching and learning and a threat to the authenticity of assessment. We have highlighted evidence that GenAI tools can improve productivity, but not necessarily across all the skills associated with coding and that there maybe risks associated with students becoming dependent on GenAI tools. In particular, we highlight the roles that GenAI tools can play that support teaching and learning, namely as a topic expert, coding partner, and puzzle setter. We also explore how to preserve authenticity in assessment in the age of GenAI, where educators must combine several deliberate strategies suggested in this chapter. Strengthened supervision can help ensure genuine student performance, while designing assignments that integrate learning from multiple sources and modalities — such as text, videos, and external materials — increases the cognitive demand and reduces the ease of GenAI misuse. Techniques such as embedding hidden text, using reverse-engineering-based tasks, and working with unsighted or bespoke datasets make it harder for GenAI tools to generate correct answers without genuine student engagement. Finally, incorporating structured student self-assessment encourages transparency, reflection, and responsible use of GenAI, supporting more authentic demonstration of learning.

Ultimately, we believe that the introduction of GenAI will transform coding so that the coder's role becomes that of a code manager. This may change the approach taken to the coding exercises in courses. However, these transformations are so central to how coding is taught, learned, and applied that not all students may be able to adapt equally well across all ability levels. Hence, there is a significant risk that their introduction may exacerbate the gap in abilities rather than close it.

References

Bente, Stefan, Natasha Randall, and Dennis Wäckerle. 2024. "A Conceptual Framework to Transform Coding Education in Times of Generative AI." In *Software Engineering Im Unterricht Der Hochschulen 2024*, 93–104. Gesellschaft für Informatik eV. https://doi.org/10.18420/seuh2024_08.

Bhatia, Shreya, Tarushi Gandhi, Dhruv Kumar, and Pankaj Jalote. 2023. "Unit Test Generation Using Generative AI: A Comparative Performance Analysis of Autogeneration Tools." In *Proceedings of the 2024 IEEE/ACM International Workshop on Large Language Models for Code (LLM4Code)*, 54–61. https://doi.org/10.1145/3643795.3648396.

Boers, Jelle, Terra Etty, Martine Baars, and Kim van Broekhoven. 2025. "Exploring Cognitive Strategies in Human-AI Interaction: ChatGPT's Role in Creative Tasks." *Journal of Creativity* 35 (1): 100095. https://doi.org/10.1016/j.yjoc.2025.100095.

Bouzid, Sara, and Loïs Piron. 2024. "Leveraging Generative AI in Short Document Indexing." *Electronics* 13 (17): 3563. https://doi.org/10.3390/electronics13173563.

Brennan, Robert W, and Brenda McDermott. 2023. "Coding Literacy in the Age of Generative Ai." In *International Workshop on Service Orientation in Holonic and Multi-Agent Manufacturing*, 445–56. Springer. https://doi.org/10.1007/978-3-031-53445-4_37.

Bughin, Jacques. 2024. "The Role of Firm AI Capabilities in Generative AI-Pair Coding." *Journal of Decision Systems*, 1–22. https://doi.org/10.1080/12460125.2024.2428187.

Chen, Tianyi. 2024. "The Impact of Ai-Pair Programmers on Code Quality and Developer Satisfaction: Evidence from Timi Studio." In *Proceedings of the 2024 International Conference on Generative Artificial Intelligence and Information Security*, 201–5. ACM. https://doi.org/10.1145/3665348.3665383.

Combrinck, Celeste, and Nelé Loubser. 2025. "Student Self-Reflection as a Tool for Managing GenAI Use in Large Class Assessment." *Discover Education* 4 (1): 72. https://doi.org/10.1007/s44217-025-00461-2.

Cotelli Kureth, Sara, Elisabeth Paliot, and Suzana Zink. 2025. "Fostering Transparency: A Critical Introduction of Generative AI in Students' Assignments." *Language Learning in Higher Education* 15 (1): 63–85. https://doi.org/10.1515/cercles-2024-0105.

Coutinho, Mariana, Lorena Marques, Anderson Santos, Marcio Dahia, Cesar França, and Ronnie de Souza Santos. 2024. "The Role of Generative Ai in Software Development Productivity: A Pilot Case Study." In *Proceedings of the 1st ACM International Conference on AI-Powered Software*, 131–38. ACM. https://doi.org/10.1145/3664646.3664773.

Daniel, Kangwa, Msafiri Mgambi Msambwa, and Zhang Wen. 2025. "Can Generative AI Revolutionise Academic Skills Development in Higher Education? A Systematic Literature Review." *European Journal of Education* 60 (1): e70036. https://doi.org/10.1111/ejed.70036.

Fan, Yizhou, Luzhen Tang, Huixiao Le, Kejie Shen, Shufang Tan, Yueying Zhao, Yuan Shen, Xinyu Li, and Dragan Gašević. 2024. "Beware of Metacognitive Laziness: Effects of Generative Artificial Intelligence on Learning Motivation, Processes, and Performance." *British Journal of Educational Technology* 56 (2): 489–530. https://doi.org/10.1111/bjet.13544.

Gan, Chengguang, and Tatsunori Mori. 2023. "Sensitivity and Robustness of Large Language Models to Prompt Template in Japanese Text Classification Tasks." In *Proceedings of the 37th Pacific Asia Conference on Language, Information and Computation*, 1–11.

Heung, Yuk Mui Elly, and Thomas KF Chiu. 2025. “How ChatGPT Impacts Student Engagement from a Systematic Review and Meta-Analysis Study.” *Computers and Education: Artificial Intelligence* 8: 100361. https://doi.org/10.1016/j.caeai.2025.100361.

Hou, Xinying, Zihan Wu, Xu Wang, and Barbara J Ericson. 2024. “Codetailor: Llm-Powered Personalized Parsons Puzzles for Engaging Support While Learning Programming.” In *Proceedings of the Eleventh ACM Conference on Learning Scale*, 51–62. ACM. https://doi.org/10.1145/3657604.3662032.

Hua, Haowei, and Co-Jiayu Yao. 2024. “Investigating Generative AI Models and Detection Techniques: Impacts of Tokenization and Dataset Size on Identification of AI-Generated Text.” *Frontiers in Artificial Intelligence* 7: 1469197. https://doi.org/10.3389/frai.2024.1469197.

Imai, Saki. 2022. “Is Github Copilot a Substitute for Human Pair-Programming? An Empirical Study.” In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*, 319–21. IEEE. https://doi.org/10.1145/3510454.3522684.

Ji, Yu, Zehui Zhan, Tingting Li, Xuanxuan Zou, and Siyuan Lyu. 2025. “Human-Machine Co-Creation: The Effects of ChatGPT on Students’ Learning Performance, AI Awareness, Critical Thinking, and Cognitive Load in a STEM Course Towards Entrepreneurship.” *IEEE Transactions on Learning Technologies* 18: 402–15. https://doi.org/10.1109/TLT.2025.3554584.

Jiang, Zhengbao, Frank F Xu, Jun Araki, and Graham Neubig. 2019. “How Can We Know What Language Models Know?” *Transactions of the Association for Computational Linguistics* 8: 423–38. https://doi.org/10.1162/tacl_a_00324.

Kanont, Kraisila, Pawarit Pingmuang, Thewawuth Simasathien, Suchaya Wisnuwong, Benz Wiwatsiripong, Kanitta Poonpirome, Noawanit Songkram, and Jintavee Khlaisang. 2024. “Generative-AI, a Learning Assistant? Factors Influencing Higher-Ed Students’ Technology Acceptance.” *Electronic Journal of e-Learning* 22 (6): 18–33. https://doi.org/10.34190/ejel.22.6.3196.

Kosmyna, Nataliya, Eugene Hauptmann, Ye Tong Yuan, Jessica Situ, Xian-Hao Liao, Ashly Vivian Beresnitzky, Iris Braunstein, and Pattie Maes. 2025. “Your Brain on ChatGPT: Accumulation of Cognitive Debt When Using an AI Assistant for Essay Writing Task.” *arXiv.org* 4.

Lyu, Wenhan, Yimeng Wang, Yifan Sun, and Yixuan Zhang. 2025. “Will Your Next Pair Programming Partner Be Human? An Empirical Evaluation of Generative Ai as a Collaborative Teammate in a Semester-Long Classroom Setting.” In *Proceedings of the Twelfth ACM Conference on Learning Scale*, 83–94. ACM. https://doi.org/10.1145/3698205.3729544.

Miao, Jing, Charat Thongprayoon, Supawadee Suppadungsuk, Oscar A Garcia Valencia, and Wisit Cheungpasitporn. 2024. “Integrating Retrieval-Augmented Generation with Large Language Models in Nephrology: Advancing Practical Applications.” *Medicina* 60 (3): 445. https://doi.org/10.3390/medicina60030445.

Mock, Moritz, Jorge Melegati, and Barbara Russo. 2024. “Generative AI for Test Driven Development: Preliminary Results.” In *International Conference on Agile Software Development*, 24–32. Springer. https://doi.org/10.1007/978-3-031-72781-8_3.

Northwood, Chris. 2018. *The Full Stack Developer: Your Essential Guide to the Everyday Skills Expected of a Modern Full Stack Web Developer*. Springer.

Nyaaba, Matthew, Lehong Shi, Macharious Nabang, Xiaoming Zhai, Patrick Kyeremeh, Samuel Arthur Ayoberd, and Bismark Nyaaba Akanzire. 2024. “Generative AI as a Learning Buddy and Teaching Assistant: Pre-Service Teachers’ Uses and Attitudes.” *arXiv.org*. https://doi.org/10.48550/arXiv.2407.11983.

Osherove, Roy. 2009. “The Art of Unit Testing: With Examples In.” *NET Manning*.

Pearson, Helen. 2025. “Are the Internet and AI Affecting Our Memory.” *Nature* 638 (8049): 26–28. https://doi.org/10.1038/d41586-025-00292-z.

Ponmalar, P Sridevi et al. 2025. “Evaluation Framework for Large Language Models: Assessing Factual Consistency and Answer Relevancy in Question-Answering Tasks.” In *2025 2nd International Conference on Research Methodologies in Knowledge Management, Artificial Intelligence and Telecommunication Engineering (RMKMATE)*, 1–6. IEEE; IEEE. https://doi.org/10.1109/RMKMATE64874.2025.11042605.

Pourcel, Julien, Cédric Colas, Gaia Molinaro, Pierre-Yves Oudeyer, and Laetitia Teodorescu. 2024. “Aces: Generating a Diversity of Challenging Programming Puzzles with Autotelic Generative Models.” *Neural Information Processing Systems* 37: 67627–62. https://doi.org/10.52202/079017-2160.

Raju, Bondu. 2025. “Generative AI as a Teaching Assistant: Opportunities and Challenges.” *International Journal of Integrated Research and Practice* 1: 28–36. https://doi.org/10.25215/31075037.031.

Rottner, Renee, Lenore Porter, Jason Bock, Jordan Jannone, Rory Walsh Senerchia, Janet Ward, and Joshuah Whittinghill. 2025. “AI and the Digital Divide.” In *Teaching and Learning in the Age of Generative AI*, 309–31. Routledge. https://doi.org/10.4324/9781032688602-19.

Suárez, David, and Begoña García-Mariñoso. 2025. “On the Verge of a Digital Divide in the Use of Generative AI?” *Telecommunications Policy* 49 (7): 102997. https://doi.org/10.1016/j.telpol.2025.102997.

Trautsch, Fabian, and Jens Grabowski. 2017. “Are There Any Unit Tests? An Empirical Study on Unit Testing in Open Source Python Projects.” In *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, 207–18. IEEE; IEEE. https://doi.org/10.1109/ICST.2017.26.

Vorel, Roman. 2025. “Generative AI for Coding and Unit Testing.” In *NoOps: How AI Agents Are Reinventing DevOps and Software*, 105–18. Apress. https://doi.org/10.1007/979-8-8688-1694-9_6.

Watterson, Steven, Sarah Atkinson, Elaine Murray, and Andrew McDowell. 2025. “AI as a Teaching Tool and Learning Partner.” *arXiv.org*. https://doi.org/10.48550/arXiv.2509.13899.

AI, Voice, and Style in Programming Education

John Wilson

Edinburgh Medical School, University of Edinburgh
jwilso3@ed.ac.uk

This article originally appeared as a post on the University of Edinburgh blog Teaching Matters in February 2025.

Back in September 2024, an "Angry Developer" posted their opinion on the effect of AI code tools (Why Copilot is making programmers worse at programming[46]).

This got me thinking about the effect of AI LLM tools on *academic skills*. Hopefully in a way that gives some useful perspective on why-or-when *not* to turn to AI, beyond the "because the uni said 'No!'" rules, and that might resonate with our students on a more personal level.

Erosion of Academic Skills

Just as AI can cause programmers to lose their deep understanding of coding principles, it can also cause every one of us to lose essential academic skills. An important reason for this is that LLMs have **no knowledge of their own!** They *literally* make stuff up (Hicks, Humphries, and Slater 2024). Therefore, relying too heavily on AI to write reports will have an impact on students' development of critical reasoning and writing skills. If students mostly use AI to write their work, there will be a noticeable (negative) effect on how they learn to structure arguments and use evidence to effectively support their narrative. Similarly, if students rely on AI to generate code, they are likely to lose opportunities to develop the important underlying skills like problem-solving and debugging skills. Over time this may reduce their ability to explain their thinking, evaluate alternative approaches, and adapt solutions independently.

As a marker, one of the things that really stands out about AI-generated work is the "voice", or in programming, probably best compared to "style". By relying on AI, students not only remove their voice from the content, but they lose the opportunity of ever even developing their own voice in the first instance. And as coders they produce code that maybe works or is technically fine, but feels disjoint, somehow disconnected from the wider purpose and context of their project.

Reduced Practical Learning

In programming, we never get our code right first time, and our feedback comes in the form of error messages. In our own studies, engaging directly with material is essential for "deep learning" (pun not *really* intended, but I'll take it). For example, if students routinely use AI to quickly summarise articles or generate lab reports, they will find it more difficult to develop a whole chain of skills.

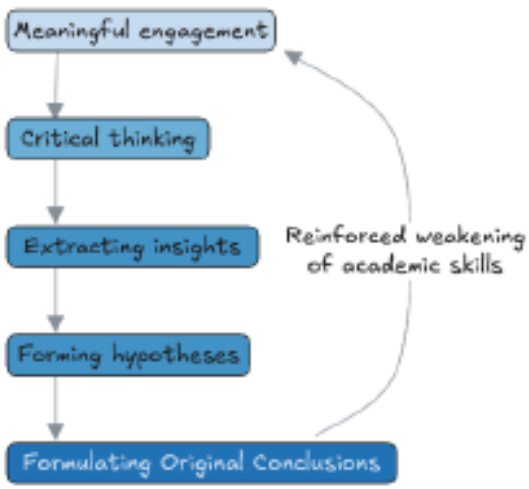


Figure 10.1: Over-reliance on AI tools is likely to weaken key academic skills, which once weakened, will lead to a cycle of continued skills impoverishment. (Totally based on opinion - I got no refs for this)

[46]https://www.darrenhorrocks.co.uk/why-copilot-making-programmers-worse-at-programming/

And most importantly, if the work is mostly AI-generated, the feedback the student receives does not apply to them personally. So it's worthless!

Dependence on AI for Solutions

AI generated code often results in code that is… just… a bit… *odd!* And that is assuming that the code even works. So, when programmers use AI to generate code, they might not fully understand the solutions it has provided. There is a parallel here in academic settings where students might use AI to complete assignments without fully engaging with the underlying concepts. I think it should be self-evident that copy-pasting a generated response does nothing to help us as academics. But it flows beyond just the academic context. Imagine a situation where you were asked a coding question identical to one you encountered in a University exam… in a *job interview*. But imagine you had used AI to produce your exam answer…?

Narrowed Creative Thinking

AI tools can definitely be an efficient way to complete a task, but they might limit creative thinking. In academic work, students might miss out on the creative process of developing unique (think "novel") ideas and solutions. Yes, AI might sometimes suggest a promising idea, but it really does not have any clue about the topic. And, with an over-reliance on AI for our work, neither do we. For example, using AI to brainstorm essay questions or projects can prevent students from both conceiving and exploring their own interesting ideas. Again, if it helps, think in terms of "novelty", often a requirement for publishing later in their careers.

Promoting Students' Lived Experience, Individuality and Own Voice

AI models are trained on vast datasets from the internet, reflecting the "most common" values and perspectives, often dominated by only a few societal groups. This can lead to reinforcement of their viewpoints while under representing diverse experiences, especially those from minority groups.

Relying heavily on AI for ideas might dilute students' unique lived experience, regardless of their background. Remember, AI lacks personal insights and creativity. Helping students understand that having their own voice shine through in their work, will become an increasingly important approach to showing independence and judgement in an AI world. Because in learning and teaching this is what distinguishes students' work from that of others, student or AI. Students may need to be shown how to critically assess AI-generated content, particularly when they have unique perspectives to share. As educators we need to highlight to students that their individuality, lived experiences, and perspectives have value, but are in danger of being lost in an AI-generated assignment.

In programming, these individual perspectives often show through in the way students interpret requirements. For example how they frame problems, or identify user needs. By helping students see that their final choice of a solution might be informed by their own, personal context we can, perhaps, help them understand why AI generated code sometimes just misses the mark.

Balancing AI Use

While AI can be a powerful tool to enhance learning and productivity, it's important for you to use it as a supplement rather than a replacement for your own efforts. Encouraging a balanced approach can help maintain and even enhance your critical thinking skills. For example, you can use AI to check your work or, perhaps, generate ideas, but make sure to engage deeply with the material and develop your own understanding and solutions. If you need any further convincing, ask yourself if you have ever felt "imposter syndrome". Then, ask if the confidence with which AI speaks actually increases your own confidence in your own knowledge…

AI tools like Copilot certainly offer significant benefits, but what we need to impress upon students is that using these tools has a real effect on development of vital academic skills. Even using for the widely "accepted" use cases, e.g. "idea generation", is problematic. If ideas-generation or direction-finding are delegated to AI, students risk becoming invisible in their own work, their work risks disappearing from the wider literature landscape, and, ultimately, they end up doing the bidding of our robot overlords.

References

Hicks, Michael Townsen, James Humphries, and Joe Slater. 2024. "ChatGPT Is Bullshit." *Ethics and Information Technology* 26 (2): 1–10. https://doi.org/10.1007/s10676-024-09775-5.

How Teaching Programming Across Disciplines Can Instil Systems Thinking

Leo Riviera

Voxa
leo.riviera@voxa.dev

Maeve Murphy Quinlan

Research Computing, University of Leeds
m.murphyquinlan@leeds.ac.uk

Rebecca L. Colquhoun

Department of Earth Science and Engineering, Imperial College London

Devanjan Bhattacharya

WARSHARE, School of Social and Political Science, University of Edinburgh; MagIC, Nova Information Management School, Universidade Nova de Lisboa
d.bhattacharya@ed.ac.uk

Introduction

The purpose of university has been the subject of much discussion (Gupta 2021). Some see it as a focal point for the humanist tradition in the Renaissance fashion, where one goes to engage with and anchor themselves in the great work of man. Others see it as a site to acquire knowledge, skills, values, beliefs, and commitments; young people develop their mental faculties and define their world views, becoming more ready to integrate into society (McArthur 2011).

Some see it as a government-mandated institution crucial to the smooth function of certain specialist fields. A university ensures, through rigorous academic examination, that the doctor treating your illness, the solicitor defending your case, and the architect designing your home are all sufficiently competent (Ryan, Toohey, and Hughes 1996).

Universities function as hotbeds of cutting-edge research, funded by subsidies from the public purse and investment from industry. Given the market-oriented university system of the UK and its ruthless focus on “value for money” (Wilkinson and Wilkinson 2020; McArthur 2011), some posit the core focus of higher education institutions to be to produce findings with some practical application. We, of course, cannot forget about the university as a modifier of social status.

However, it is impossible to deny that one of the core functions of a university is to educate. Universities provide their students with the knowledge necessary to pursue their chosen career paths, and provide a space for individuals to develop and hone their chosen skill. When viewed in this light, the value of teaching programming across disciplines becomes apparent.

Teaching Computational Thinking Through Programming

We have long foreseen a world where computers are ubiquitous; today, they are undeniably present in almost every aspect of modern life (Weiser 1999; Stewart 2012). Over the past few decades, they have become fundamental to the way we communicate, work, analyze data, and solve complex problems. Understanding how to use a computer is no longer a skill reserved for computer scientists, but serves as an invaluable tool for problem-solving across all disciplines. In fact, the skill is not only crucial to being a good worker, but also a good citizen who can affect change. With some of the opinion that computer literacy in decline among young people (Stewart 2012), teaching programming across disciplines can provide students with the base knowledge they need to understand the computers they come across during their careers.

Learning to code is associated with improved problem solving skills in some domains (Scherer, Siddiq, and Sánchez-Scherer 2021). When we teach students to program, we often start by introducing them to a language’s syntax. However, we also introduce them to resources for self-study, like online tutorials. We teach how to find useful resources and evaluate their quality. In their self-study, students discover reference documentation and language specifications. That, in turn, supports their discovery of capabilities of the programming language they were never formally taught. Students eventually learn not only how to write code but also how to teach themselves.

In learning to code, students encounter computational thinking (Jeannette M. Wing 2006). Computational thinking involves taking a problem and developing solutions we can use a computer to execute. Although different models of computational thinking consist of different steps, Shute, Sun,

and Asbell-Clarke (2017) describes it as a process which includes

- Problem reformulation, where we reframe a problem into a solvable and familiar one
- Recursion, where we incrementally construct a solution based on preceding information
- Decomposition, where we break a large problem down into more manageable sub-problems
- Abstraction, where we model the core aspects of a complex problem or system
- Systematic testing, where we take purposeful steps to derive solutions

Computational thinking has applications beyond computing. It is seen as fundamental for analysing and solving all sorts of problems, because we can transfer techniques we use to make our solutions efficient for computers to other disciplines[47] (Jeannette M. Wing 2010, 2012). Programming is a tool to teach computational thinking, but computational thinking can be applied to other problems. Even if one disagrees with the premise that computational thinking provides skills which are directly transferrable to other disciplines, learning to code is a good form of mental exercise and teaches us to solve problems (Love 2019; Scherer, Siddiq, and Sánchez-Scherer 2021).

However, teaching programming outside of computer science courses may have another, less obvious benefit: instilling systems thinking skills in students. To understand why this could have value, we need to take a brief foray into the world of systems.

A Leap Into Systems Thinking

A system is a collection of entities which are connected to each other and work together for some purpose. As a result of interactions between entities, the system produces its own pattern of behaviour over time. The pattern of behaviour that emerges from a system can be quite different from the behaviour of its constituent parts (Kim 1999, 2–5). The process of identifying, understanding, decomposing and rebuilding systems is called systems thinking. More formally, systems thinking can be defined as "...a set of synergistic analytic skills used to improve the capability of identifying and understanding systems, predicting their behaviors, and devising modifications to them in order to produce desired effects"[48] (Arnold and Wade 2015). Systems thinking is, at its core, a tool for allowing us to see the world differently, ask different questions and get different (often unexpected, sometimes better) answers.

What we colloquially refer to as a computer today was at one point called a computer system. The computer was the hardware used to perform calculations, while the computer system was the combination of hardware and software that could be programmed to perform a specific task ("An Introduction to Computers and Computer Systems: Session 2: 6" 2021). As devices have become more capable and more compact, this differentiation has become a point of pedantry. However, for the purposes of our discussion, this distinction is helpful. The physical components of a computer (its CPU, memory, storage drives, peripherals and the like), are unable to achieve anything by themselves. Similarly, software is useless without a device to run it on. When both are combined, we have a useful system which can take in data, process it into useful information, and output it in some usable form. We have a computer system!

By our definition of a system, computer programs are also systems. They are a collection of elements (most obviously input, source code, and stored data) connected together (in the body of a program and in a machine's memory)[49] for the purpose of producing outputs over time. Let us explore this further.

Computer Programs as Systems

The obvious elements of a simple computer program include, among other things,

- its inputs,
- its outputs,
- its application code,
- the internal state it maintains and any stored data.

Computer programs also feature many intangible elements, including

- accessibility for users of assistive technologies and users with disabilities,
- its availability and uptime,
- its maintainability, and the ease with which changes can be made,
- its correctness, where a given input produces an expected output. This can be assured with manual and

[47] The point we're making here is quite nuanced. Analysis by Scherer, Siddiq, and Sánchez-Scherer (2021) establishes that learning to code may be associated with *some* broader cognitive benefit, with any possible transfer of learning coming with an explicit cost in the form of sufficient teaching time, supportive educational environments, and teacher training. However, this chapter posits that computational thinking, often taught through computer programming, provides a cognitive framework that makes solving problems in other disciplines easier.

[48] Interestingly, Arnold and Wade (2015) take a systems thinking approach to defining systems thinking. How meta!

[49] The theoretical Turing machine springs to mind, here. Despite its simplicity, it forms the basis for how many computer scientists think about computation today.

automated system testing, and in some cases, can be formally verified,

- its stability, in its ability to handle large volumes of input and remain stable if those limits are exceeded,
- the aesthetics of its source code, often enforced by code review and code linters.

These elements are inherently interconnected and dependent on each other. The application code transforms the input, considering and possibly updating its internal state in the process. Data can be stored for retrieval at another time, and can form a state which affects future outputs of the same program. The application code directly affects the correctness and stability of the program. A computer program produces output to fulfil its purpose, and that output can even be the input of another computer program. The general purpose of computer programs is to compute, to process some form of input into some form of output. The purpose of a particular piece of software can be deduced by how it processes data, looking at its inputs, outputs, and side effects. This makes a computer program a system which, in turn, runs on a computer system. It's a system within a system!

Not all elements matter equally when designing a program. It depends on the purpose. Scripts for calculating the answers to mathematical problems on Project Euler[50][51] would emphasise correctness. However, a Government website designed to provide information to the public would emphasise accessibility and availability (Government Digital Service 2012). An online multiplayer video game would likely focus on maintainability to roll out changes quickly, and stability to handle spikes in traffic from players. A computer program written to control a nuclear power plant looks very different from one used to track library books!

Like any other system, a software program's response to input is characteristic of itself[52]. It depends on the way we have encoded instructions to process input and produce output. This is even more true for a program which holds state or uses randomness, given the same inputs may produce drastically different results based on some internal state. More complex programs can even change and adapt their behaviour in response to their environment and their input. For example, a rate-limiting system can restrict requests from a user to prevent overload. Programs can also be fault-tolerant, using circuit breakers to avoid repeatedly trying to run an operation which fails. These can be considered balancing feedback loops, intended to prevent failure and keep performance metrics within some desired threshold.

In the real world, it is more likely that a computer program that is actually useful is doing more than one thing. Like many complex systems, computer programs can be composed of subsystems. Although most of a program's behaviours are explicitly designed and defined in application code, many arise as a logical consequence of different elements interacting with one another. This emergent behaviour is another key characteristic of systems (Meadows, n.d., 11–34).

Thinking one level up from a program, computer systems are made up of several programs that are constantly interacting with each other; a program itself forms a subsystem within a computer system. The Unix kernel schedules time for programs, background processes and system operations, each of which affects its environment and is affected by its environment (Alomari, n.d.). A program can affect its execution environment (for example, by storing data on disk or holding it in memory), and the environment can affect the program (by interrupting its normal execution). Feedback loops like these are common in systems found in the real world.

As people interact with the world, they intuitively build up an understanding of and appreciation for complex systems (Forrester 1971). Teaching students programming merely provides them with the language to describe and interact with complex systems. This knowledge can then be used to understand other systems which make up the world. Of course, there are several concepts in systems thinking which have no obvious analogies when discussing computer programs. However, the way we build, use and debug computer programs provides a useful introduction to systems thinking.

Much like any other system, a computer program's purpose is what it does, regardless of the creator's intent (Meadows, n.d.). Depending on how a program behaves within and interacts with its execution environment, interesting (and sometimes dangerous) side effects may occur. To find and fix unexpected behaviours in our computer programs, we carefully examine the individual components and how they work together, in the context of the desired function. Similarly, complex systems have structures which contain latent behaviours. In fact, a big challenge of systems thinking is to arrange the structures and conditions of a system to encourage and amplify beneficial behaviours, and limit and reduce destructive behaviours (Meadows, n.d., 72).

With a computer program, we may have access to logging, tracing, and observability tools which help us diagnose deviant behaviours. Eventually, we identify the element or interconnect which has caused our unexpected behaviours (colloquially known as a "bug"), and we make an adjustment to remediate it. The methodical troubleshooting approach can be applied to the systems which

[50] https://projecteuler.net/

[51] Project Euler is a collection of 981 (at time of writing) mathematical problems designed to be solved with computer programs. It allows users to discover and discuss solutions to these problems, and is quite a popular resource for learning programming.

[52] Deterministic programs always produce the same output for a given input. Non-deterministic programs can produce different outputs for the same input, depending on factors such as internal state, randomness, or concurrency. Regardless of whether a program is deterministic or non-deterministic, its behaviour can be understood and predicted based on its design and implementation.

govern the behaviour of friend groups, companies, charities, universities, market economies, and other groups of interconnected entities.

Decomposing Conceptual Systems With Computational Thinking

We can decompose the systems which exist in our world using the same process we use to construct computer programs: computational thinking. As mentioned previously, the end result of the computational thinking process is algorithms that can solve a problem. Along with other components, the algorithms we define and the relationships between them form a system we call a computer program. Perhaps performing the steps of computational thinking in reverse could be useful for decomposing a system into its constituent parts?

In his *Introduction to Systems Thinking*, Kim (1999, 17–18) details the different modes we can act in when dealing with a system. Thinking about these modes, particularly within the context of computational thinking, allows us to transcend from being acted on by a system to acting on a system.

Our interactions with systems are normally restricted to reacting to what it does. After observing enough of its behaviour, we can notice patterns in those events we can adapt to. A system's architecture, also referred to as a "systemic structure", is the way a system's components are organised. It is responsible for the events and patterns we observe (Kim 1999).

When examining a system's architecture, we can start with systematic testing from computational thinking. We can ask ourselves: what steps could someone have taken to determine the elements that make up this system, and how they are connected to each other? Where could they have searched? Who could they have asked for a review?

Next, considering its stated purpose, we can de-abstract the system. If we were designing a system to achieve a similar goal, what detail would I consider irrelevant? Putting myself in the shoes of the people who have built and maintained this system, what are their incentives? What are this system's intended goals, versus its real impact? Is there any detail about the real world that, when abstracted, would cause the system to act in the way it does? Is there a way to improve this?

Considering the system as a whole, we can recursively recompose it. Looking at its purpose and emergent behaviour, what individual components make up the system? How could those be linked to one another? Based on the model I've constructed, what about this system could be broken? What about it could be inefficient? What about its approach to problem solving is inelegant? Does it have any reason to be like that?

Finally, we can ask ourselves: what is this system trying to do, and what was the intent? What output does it produce? What side effects can be directly attributed to it? How does its output affect other systems? What adjacent problems exist that this could be tweaked to solve? What is the real purpose of this system?

A system is a useful abstraction for thinking about behaviour which emerges from the interactions between different components. However, as it is an abstraction, the systems we concoct using this method will never be complete. There will always be context we lack and so we will omit components and connections between them. However, interrogating a system in this way could prove useful depending on the problem we're trying to solve.

When picking apart a system in this way, we look at things it does and look at trends in the things it does. Eventually, we can consider the way the system and its structures are organised to produce patterns which then produce outcomes. We can then consider what components should make up a system, and how they ought to be organised. This is similar to how we're able to make changes to a program's implementation to improve it or change its behaviour once we understand what it does and how it does it.

Computer programs differ greatly from systems found in the real world, because we can change and understand computer programs comparatively easily. They have defined execution environments and can be disabled if a feedback loop causes undesirable consequences. However, despite the fact computer programs are relatively simple systems, they are systems nevertheless. This means computational thinking is useful to identify systems, and develop our capacity to be more creative, reflective and generative when dealing with systems.

Rebuilding the World in Our Image

During the process of designing systems, we can consider our mental model of the world. Reflecting on how systems are designed, as well as how we would design them, exposes the assumptions we harbour about how the world works. We can then imagine new ways of organising a system's components. Understanding and changing the way we model the world in our heads allows us to usher into existence entirely new systems. Systems thinking allows us to constantly question how the world works, and what is actually important. From there, it is not a large leap to think about the world we want[53].

Our thinking is bounded by our ability to describe the world. If we lack the

[53] There are interesting parallels to be drawn with Derrida's deconstruction, but this is left as an exercise to the reader.

language to discuss change, we will struggle to conceptualise it. Put more succinctly by Ludwig Wittgenstein, the limits of one's language are the limits of their world (Wittgenstein, n.d.). Teaching programming instils computational thinking, as a stepping stone to systems thinking. With systems thinking, we have a language to describe the way the world works and how we can change it. In fact, the two work in tandem. Computational thinking provides us the ability to solve complex problems by breaking them down into smaller parts; systems thinking gives us a way to model how those smaller parts fit together and interact with one another.

What we teach speaks to the kind of people we want in the world. The type of work we do influences the type of people we are. We grow to fit the jobs we work based on the skills we use every day (Woods et al. 2020). We should encourage our students to develop systems thinking skills through programming, and to use those skills as part of their problem-solving toolkit in the workplace. Equipping people with the ability to identify, decompose and eventually restructure the systems which govern our world will make them more effective, more independent, and more capable agents of change. They will finally see the world for what it is: a collection of elements to be examined, interrogated, and reconstructed in the image of something more equitable and more sustainable.

Acknowledgements

Thank you to everyone who has offered feedback on this chapter, including Dr. Mark Stonehouse, Dr. Heather Allansdottir, Ole Thal, and other entities who wish to remain anonymous. Your input has been invaluable.

References

Alomari, Ahmed. n.d. *Oracle 8i and UNIX Performance Tuning*. The Prentice Hall PTR Oracle Series. Prentice Hall.

"An Introduction to Computers and Computer Systems: Session 2: 6." 2021. The Open University. https://www.open.edu/openlearn/mod/oucontent/view.php?id=109019&section=6.

Arnold, Ross D., and Jon P. Wade. 2015. "A Definition of Systems Thinking: A Systems Approach." *Procedia Computer Science* 44: 669–78. https://doi.org/10.1016/j.procs.2015.03.050.

Forrester, Jay W. 1971. "Counterintuitive Behavior of Social Systems." *Technological Forecasting and Social Change* 3: 1–22. https://doi.org/10.1016/s0040-1625(71)80001-x.

Government Digital Service. 2012. "Government Design Principles." GOV.UK; https://www.gov.uk/guidance/government-design-principles#this-is-for-everyone.

Gupta, Achala. 2021. "What's the Purpose of University? Your Answer May Depend on How Much It Costs You - LSE Impact." London School of Economics; Political Science. https://blogs.lse.ac.uk/impactofsocialsciences/2021/01/21/whats-the-purpose-of-university-your-answer-may-depend-on-how-much-it-costs-you/.

Kim, Daniel H. 1999. *Introduction to Systems Thinking*. Pegasus Communications.

Love, David. 2019. "Computational Thinking – a New Modality of Thought or Just What Coders Do?" https://digiteacher.wordpress.com/2019/09/20/computational-thinking-a-new-modality-of-thought-or-just-what-coders-do/.

McArthur, Jan. 2011. "Reconsidering the Social and Economic Purposes of Higher Education." *Higher Education*

Research & Development 30 (6): 737–49. https://doi.org/10.1080/07294360.2010.539596.

Meadows, Donella H. n.d. *Thinking in Systems*. Chelsea Green Publishing.

Ryan, Greg, Susan Toohey, and Chris Hughes. 1996. “The Purpose, Value and Structure of the Practicum in Higher Education: A Literature Review.” *Higher Education* 31 (3): 355–77. https://doi.org/10.1007/bf00128437.

Scherer, Ronny, Fazilat Siddiq, and Bárbara Sánchez-Scherer. 2021. “Some Evidence on the Cognitive Benefits of Learning to Code.” *Frontiers in Psychology* 12. https://doi.org/10.3389/fpsyg.2021.559424.

Shute, Valerie J., Chen Sun, and Jodi Asbell-Clarke. 2017. “Demystifying Computational Thinking.” *Educational Research Review* 22: 142–58. https://doi.org/10.1016/j.edurev.2017.09.003.

Stewart, Tom. 2012. “Computers Are Everywhere.” *Behaviour & Information Technology* 31 (4). https://doi.org/10.1080/0144929x.2012.675162.

Weiser, Mark. 1999. “The Computer for the 21st Century.” *ACM SIGMOBILE Mobile Computing and Communications Review* 3 (3): 3–11. https://doi.org/10.1145/329124.329126.

Wilkinson, L. C., and M. D. Wilkinson. 2020. “Value for Money and the Commodification of Higher Education: Front-Line Narratives.” *Teaching in Higher Education* 28 (2). https://doi.org/10.1080/13562517.2020.1819226.

Wing, Jeannette M. 2006. “Computational Thinking.” *Communications of the ACM* 49 (3): 33–35.

Wing, Jeannette M. 2010. “Computational Thinking: What and Why?” https://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf.

———. 2012. “Computational Thinking.” https://www.microsoft.com/en-us/research/wp-content/uploads/2012/08/Jeannette_Wing.pdf.

Wittgenstein, Ludwig. n.d. *Tractatus Logico-Philosophicus*. 2nd ed. Routledge Classics. Routledge.

Woods, Stephen A., Grant W. Edmonds, Sarah E. Hampson, and Filip Lievens. 2020. “How Our Work Influences Who We Are: Testing a Theory of Vocational and Personality Development over Fifty Years.” *Journal of Research in Personality* 85: 103930. https://doi.org/10.1016/j.jrp.2020.103930.

Computer Says No: Reducing Technical Barriers to Help Novice Programmers

Chris Cooling

Early Career Researcher Institute, Imperial College London
c.cooling10@imperial.ac.uk

David Cutting

School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfast
d.cutting@qub.ac.uk

Ozan Evkaya

School of Mathematics and Maxwell Institute for Mathematical Sciences, University of Edinburgh
Ozan.Evkaya@ed.ac.uk

Lucia Michelin

Edinburgh Futures Institute, University of Edinburgh
lucia.michielin@ed.ac.uk

Samuel Skipsey

School of Physics and Astronomy, University of Glasgow
Samuel.Skipsey@glasgow.ac.uk

Lizzie Wadsworth

Department of Applied and Human Sciences, Kingston University
L.Wadsworth@kingston.ac.uk

Introduction

Many students come to their first programming course or workshop with little or no experience in any programming language. As well as learning how to write code, students are also required to navigate file systems, open and use interfaces, and potentially work in the terminal. There is a wide range of computer literacy amongst students. Some will possess this tacit knowledge and be completely comfortable with these tasks, but others will have to catch up with these skills, making the initial sessions much harder. Helping these students keep up with the lesson is a challenging part of teaching introductory programming courses.

If students are working on their own computers, there will be a variety of different operating systems and software configurations. This can make the provision of universally helpful instructions difficult and, in some cases, it might be difficult or impossible to install the desired tools at all. Error messages can also be difficult to decipher, particularly for novices. This can lead to students, quite reasonably, having many questions for tutors. This slows down the course, reducing what can be covered, and frustrating students who are set up and ready to go.

In this chapter, we will explore some of the techniques and tools that we, as educators, have found to help solve or mitigate these problems. Examples and observations of student behaviour are derived from our experiences in the classroom, which cover a range of class sizes from 10 to over 350 students, undergraduate foundation years to research postgraduates, across a variety of subjects from Biosciences to Physics to Computer Science.

Course Material Design

When designing a course, one of the first steps is to decide how to present teaching material to learners. In this section, we discuss approaches, mediums and file formats that can support student onboarding best.

Clear Wording, Sections, and Images

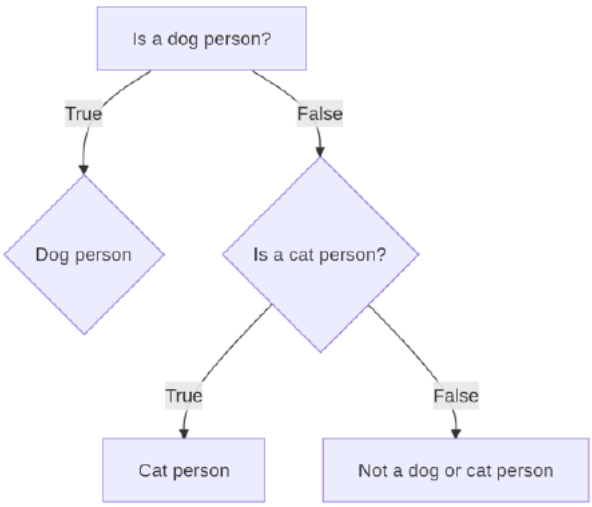
Programming for the first time can be a challenging experience for a student, as there is a lot to learn from the first class — from how to log in and set up, to how to write, build, debug and execute programs. Hence, learning material will need to contain a lot of information for students to be able to follow instructions and write their first program.

There is a lot of terminology in describing a programming language, which can confuse new students, such as variables, vectors, and functions. These are also quite abstract concepts, so finding a way to describe them in simple language can boost understanding.

Sections can help students progress through the learning material and keep track of what they have and haven't done. Clear section headings and numbering can allow students to make a note of where they are, and to tell teaching staff which section they are stuck on. Additionally, splitting up a long worksheet into several smaller pages may reduce students' feelings of overwhelm by presenting a few manageable tasks at a time.

Programming is quite text-heavy; most instructions are written in text, and code examples and output are also often text-based. Incorporating images into material can help to break this up for the students and make the material more visually stimulating. This should be done in an accessible manner, with alt-text provided where appropriate. These images could provide more explanation of a key concept (as in the image below). Alternatively, they could be images which explain the concept of the task; for example, if using a dataset which includes penguin flipper measurements, then providing an image of the represented penguin species.

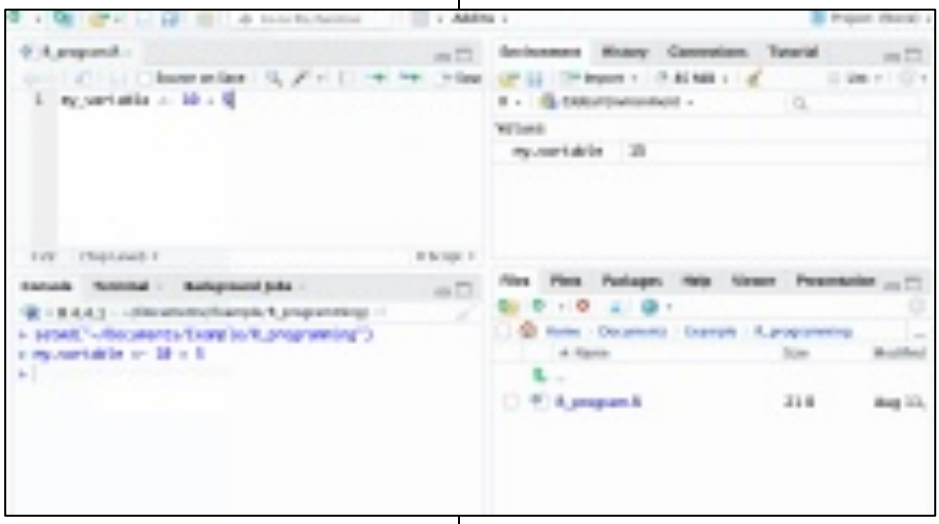
Figure 12.1: Flow chart visually demonstrating the logic behind an `if, elif, else` statement in Python, generated using the open-source JavaScript-based tool Mermaid[54].



Embedded GIF Screen Recordings

Instructions are never clear enough. You can write detailed step-by-step instructions for how to get set up, and also start a class with a live demonstration of exactly which buttons to click. You may find, however, when you start to walk around the class, that many students in an introduction to programming session have yet to run their first line of code. They may have arrived late, or had technical issues with the first computer they logged into, or missed a step in the demonstration, or for any other reason not been able to follow the instructions.

Figure 12.2: Keyframe of example GIF on how to use RStudio.



One technique which can easily enhance setup instructions is a GIF screen recording. This can show each step really clearly and works especially well with the younger generation, who are more used to video tutorials than they are following written instructions.

Figure 12.2 shows key frames from a GIF recording of how to create a new folder and an R script, along with running a line of code, in RStudio.

These can be embedded in any online-based tutorial, such as an HTML file or within a page written on a Virtual Learning Environment (VLE) such as Moodle, Blackboard or Canvas. For long processes, you can combine multiple GIFs with text instructions and clear headings (Step 1, Step 2...).

To make them, there are many different screen-recording software programs you can use. One tool is LICEcap[55], which is available for both Windows and Mac, and is free and open source. It records directly to GIF format, so it can be used straight away. Alternatives include the Snipping tool in Windows 11, which can export any screen recording of less than 30 seconds in length as a GIF, and PyPeek[56], which works on Windows, Mac and Linux environments. ScreenToGif[57], which works on Windows, allows for both MP4 and GIF creation from the same clip or GIF creation from uploaded MP4s which could be useful for making short GIFs from longer videos or screen recordings.

Using these GIFs, students can be directed to the worksheet and most will be able to follow the instructions and set themselves up, freeing up more time for practical questions on the programming language. The GIFs also become a resource they can return to, so instructions do not need to be repeated in every class.

Supplementary Materials

Some modules can require pre-existing skills in tools such as data visualisation and interpretation, and learning these skills may not be a learning outcome of the module. Instead, these skills may be assumed prior knowledge, which can cause problems for students who do not feel comfortable with these skills. For cohorts with diverse backgrounds, such as postgraduate students, extra care is needed to support novice learners. From that perspective, having clear and open communication with learners to understand their background is a good starting point. To support their learning process effectively, supplementary

[54] https://mermaid.ai/open-source/

[55] https://www.cockos.com/licecap/

[56] https://github.com/firatkiral/pypeek/

[57] https://www.screentogif.com/

learning material such as a deck of slides, glossaries and cheat sheets can be provided. These can be referenced by novice learners and can help them keep up without slowing the progress of more experienced learners.

Proactively consulting with learners can be useful to decide which type of supplementary material would be most beneficial for the cohort. A module-specific glossary of key programming terms and common troubleshooting can be a useful reference. Supplementary materials can also include summaries of technical concepts and practical code implementations. More comprehensive materials can even include worked examples or mini exercises with different styles of questions such as code completion or code adaptation.

Learners who lack some of the assumed skills may still have to work harder to complete a course, but structured and targeted supplementary materials can make it possible for these students to catch up and keep up.

One downside of the inclusion of supplementary materials is the tendency for scope creep of the course. It can be easy to take a 2 hour course and turn it into a 5 hour course with a much larger amount of content. Whilst this can provide necessary context for students with less background knowledge, it risks increasing the amount of content all students might engage with. Some students may even feel obliged to engage with the supplementary material even if it's not necessary for them. For these reasons, if supplementary materials are used, they should be minimal in covering the assumed knowledge for the course and clearly structured so students can identify which parts are relevant to them personally. Additionally, the nature of the materials as an optional resource to offer a recap of assumed knowledge should be made clear.

Pre-Recorded Videos

Pre-recorded videos can be useful, and can replace or supplement lecture-based teaching. This is known as a 'flipped classroom'. These videos can be a good way to explain complex concepts to the whole cohort, allowing students to study at their own pace when it's convenient for them. A short multiple-choice quiz after a video can be added in most VLEs and can reinforce the content of the video and help students determine whether or not they understand the concepts covered.

Students may use the videos in a variety of ways. Some students will watch the videos before class and come prepared with questions — these students tend to progress very quickly. Some students will instead watch the videos during class as they try and complete the worksheet activities. This can also be effective, as they can pause and rewind when needed (which they can't do during a live lecture). This approach can also be problematic if they are playing the video audio out loud or if they spend the whole class copying from the video and don't progress to trying any practical programming themselves.

Here are some tips for effective videos:

1. Short and concise: a series of 5-10 minute videos helps make it easier for students to maintain concentration, and can be interspersed with other activities such as readings or exercises. Shorter exercises are also easier and quicker to iterate if they need updating. If a longer video is used, clear time stamps allow students to go straight to the part they need explanation for when they hit an issue.
2. Live coding: software such as OBS[58] can be used to film both your camera and your screen so that you can demonstrate concepts by live coding. This is especially useful for demonstrating debugging — making a (purposeful) mistake in the code and then explaining the error messages and how to fix it. This could also be used to demonstrate how to search for answers on sites such as Stack Overflow, or ask an AI tool for suggestions (which may or may not be helpful!).
3. Accurate subtitles: subtitle auto-generation is continually improving, but it can be worth going through and correcting any noticeable errors. This then allows students to follow the video with subtitles in class if they don't have headphones.

Existing videos can be a good source of inspiration. There are many programming videos aimed at beginners on educational channels on YouTube. These are often high-quality videos with a high production value (polished, professional and well-made) which can be very engaging. However, not all videos need to have a production value to be useful. Videos with lower production value can be made very quickly to respond to issues raised by students during a module, or to clarify points that were unexpectedly difficult.

Embedded Executable Code Blocks

It's also possible to provide worksheets which have embedded executable code blocks. These come with the benefits of fewer installation requirements and quicker times to set up and get started, but also have some drawbacks. They can work well in combination with other methods so that students can test short bits of code without setting up elsewhere.

Jupyter (and Other) Notebooks

Jupyter Notebooks[59] can be a useful tool for learning programming, and it is possible to write a worksheet where the instructions are in a Markdown-formatted code block, followed by editable code blocks with over 40 languages supported.

One advantage of this is that everything is inline — printed outputs and graphs, for instance, can be viewed directly underneath the code block, and students can work directly on the worksheet.

Although requiring a paid subscription (a lot pricier than most other options here), CoCalc[60] is an interesting use of

[58] https://obsproject.com/

[59] https://jupyter.org/

[60] https://cocalc.com/

a Jupyter Notebook with collaborative editing. This can work quite well in a remote teaching environment. A teacher account can assign worksheets directly to student accounts, and also enter the student's folders and files. Similar to multiple people editing a shared online document, a teacher can see student edits to their work in real-time and also add notes. This feature has also been added to Noteable[61], which is hosted by EDINA at the University of Edinburgh and is available at a number of universities in the UK.

One disadvantage of Jupyter notebooks is that they are based on interactive programming, meaning that variables are kept into memory between runs of different code snippets — even if the corresponding code is then deleted. This can lead to issues when trying to debug an unusual error, where the problem code *has since been deleted*[62]. Other notebooks than Jupyter can solve this (by updating the entire notebook state on changes) such as marimo[63] for Python or Pluto[64] for Julia, but Jupyter is by far the most readily available solution.

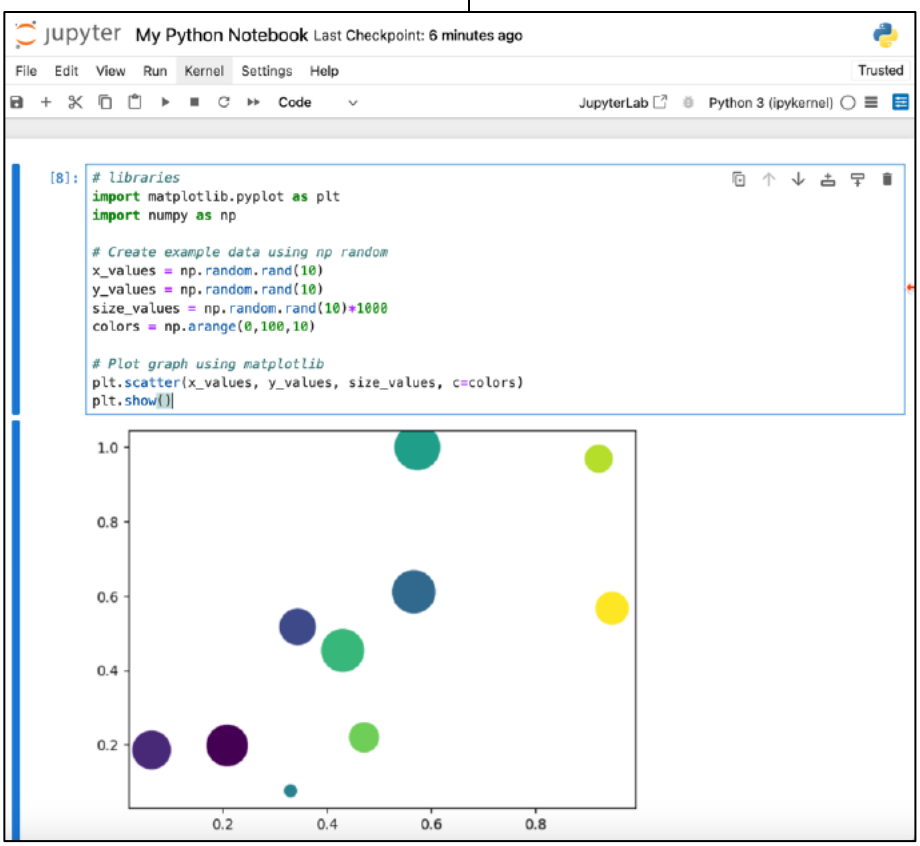


Figure 12.3: An example Jupyter notebook with a Python-generated scattergraph

Running Code in the Browser: WebR, Pyodide

WebR (for R programming) and Pyodide (for Python) allow for integration of interactive code blocks in HTML-based worksheets. If writing worksheets in Markdown using Quarto[65], the quarto-live[66] extension can be used to include these interactive code blocks.

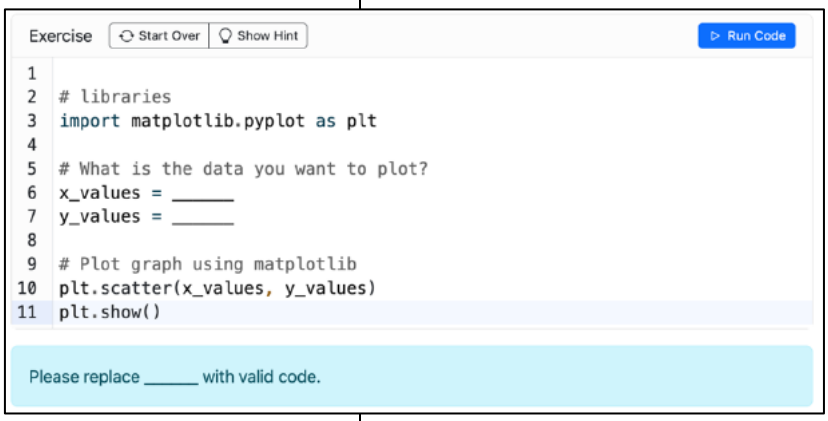


Figure 12.4: An example of a Pyodide exercise.

The advantage of these is that you can have a worksheet which prompts a student to test parts of code as they go. They can also be great for quick explanations in class, so students can test something without altering the code they are working on.

Both of these work with a variety of common packages such as `ggplot` (R) and `matplotlib` (Python). The `quarto-live` extension can also be used to generate exercises where students need to fill in the blanks in the provided code, and this can include hints and answers.

The disadvantage is that the code a student writes is not saved anywhere — so this has to be made clear to them. It is only for temporary use. Additionally, as with the Jupyter notebooks, all variables are saved into a temporary memory so one code block may function differently depending on which blocks were run previously.

[61] https://noteable.edina.ac.uk

[62] For example, one time, a student using a Jupyter Python notebook named a variable `str`, overwriting the built-in `str()` function, then deleted the code, leading to unexpected behaviour when trying to use the `str()` function later.

[63] https://marimo.io/

[64] https://plutojl.org/

[65] https://quarto.org/

[66] https://r-wasm.github.io/quarto-live/

GitHub and Other Software Development Platforms

GitHub, and its competitors such as GitLab, Forgejo, etc., are websites that allow users to store code for a project in an online repository using the Git version control system. GitHub, in particular, is currently popular and widely used in software development and data science, and provides many useful tools that we can use to make life easier for our students.

The first, simplest use of any of these sites is to store course materials in a repository. A wide variety of file types can be stored, allowing scripts (such as `.py`, `.ipynb`, `.C`) and teaching materials (such as `.md`, `.pdf` or even PowerPoint files) to be stored together. Educators can define the content of the course and its organisation into directories. Students are able to download these materials directly to their computer, making it easier to ensure students have the correct files and directory structure on their computer. Indeed, as use of version control systems themselves is a useful coding-adjacent skill, interacting with the repository can provide additional learning benefits.

Furthermore, when adopting server or Docker-based solutions in educational contexts, the collection of all educational materials in a GitHub repository markedly simplifies the process of importing data. These platforms, including Google Colab and Posit, integrate easily with GitHub, allowing effortless uploads of comprehensive educational content directly into virtual learning environments. This setup, therefore, enhances accessibility and maintains a uniform teaching approach across various platforms. Unfortunately, the integration of some such platforms with GitHub alternatives is often less good — it's often possible to get them to work with GitLab etc., but the flow is less polished.

The built-in GitHub integration streamlines the teaching workflow, enabling instructors to update teaching materials in real time. As these updates are pushed to GitHub, they can be synchronised automatically with the student's learning platforms, ensuring all participants have immediate access to the latest resources. This synchronisation eliminates the common logistical hurdles associated with outdated teaching materials or inconsistencies in software versions across different student setups.

Computing Platform

Once course materials have been created, it needs to be decided how and where students will run code as they complete courses or projects. In this section, we discuss the cloud-based options and local options, including how to support each.

Cloud-Based Development Environments

While students can see the code held on GitHub and similar platforms, they cannot run it in the browser file viewers, which can be frustrating for a novice who just wants it to simply work. Traditional local setup frequently raises barriers, often involving intricate installation and configuration steps, which might deter beginners even before they engage with the actual coding exercises. Such steps can also lead to a proliferation of setup errors, turning what should be an exciting learning adventure into a tedious, error-prone process. This issue is particularly pronounced in online and hybrid learning environments.

In fully in-person classes, instructors or teaching assistants can quickly diagnose and resolve installation problems simply by glancing at a student's screen and offering immediate guidance. In small classes, this is often manageable but can create disruption and slow down classes when many students need help. Troubleshooting in remote settings is far more complex. It often requires extended back-and-forth exchanges in the chat or moving the student into a breakout room, where they must share their screen. While screen sharing can be effective, it frequently causes students to miss important portions of the live session, creating additional learning gaps and disrupting the overall flow of instruction.

Cloud-based platforms offer an avenue to dismantle the initial barriers that commonly discourage learners or slow down live sessions. Tools like Posit and Colab allow beginner-level students to interact with code environments with a couple of clicks. These cloud-based development environments host an equivalent system on cloud-based services, often incorporating a browser-based IDE. This can help students to quickly get code running.

Cloud-based tools also offer a pre-configured, reproducible environment which requires no setup by the student. This accelerates the starting process and spares students from the demotivating sequence of 'boring' steps, allowing them to dive directly into the more gratifying aspects of programming. Moreover, some of these platforms provide free access to computing resources, which is a significant advantage, particularly for students who do not have the means to acquire powerful hardware capable of running complex code or large applications.

Cloud-based tools also bring benefits relating to accessibility and collaboration. Being cloud-based, these tools are accessible from anywhere, facilitating a flexible learning environment. Additionally, these platforms are inherently collaboration-friendly and in many cases support version control by default.

Despite these substantial benefits, several challenges persist. Pre-loaded library conflicts can arise, posing a significant hurdle compared to the more controlled, albeit time-consuming, process of manually installing each necessary library. This automated convenience could sometimes backfire, leading to intricate conflicts that are hard to resolve without comprehensive control over the server environment. Furthermore, instances of server crashes can disrupt the learning process, culminating in potentially unsaved work and progress loss.

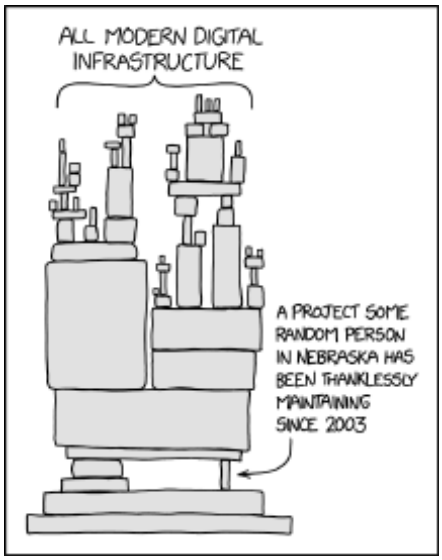


Figure 12.5: Visual representation of software dependencies, an xkcd[67] webcomic by Randall Munroe.

Admin permissions, or lack thereof, can also generate challenges, particularly when obscure errors emerge that require more than just user-level access to resolve. This limitation can frustrate users, especially when the path to a solution is barricaded by restricted permissions. Cloud-hosted development platforms can present additional challenges in remote and hybrid settings. Although the computational workload is handled on external servers, these platforms still rely heavily on the user's local machine and network stability for smooth interaction. When used concurrently with video conferencing tools required for live instruction, stresses on the user's internet connection may cause problems including lag, reduced responsiveness, or disconnection. These issues can affect instructors and learners, disrupting the continuity of teaching and learning.

Lastly, concerns around data security persist; while cloud environments offer numerous advantages, the necessity to run certain processes locally for security reasons becomes a complex task for those unfamiliar with how to configure local environments effectively.

These considerations draw a complex picture of the trade-offs involved in using cloud-based platforms for educational purposes. While they significantly lower the threshold for beginners to engage in coding by removing many logistical hurdles, they introduce a different set of challenges that need careful navigation to ensure a secure, effective, and continuous learning experience.

Codespaces

One prominent example of a cloud-based platform is the Codespace feature offered by GitHub (owned by Microsoft), which integrates Microsoft's Visual Studio Code IDE. Codespaces are a major example of a cloud-based development environment which we'll discuss in this section.

A Codespace is a development environment hosted by GitHub. Each Codespace will be based on a repository. This allows learners to visit a GitHub repository owned by the teacher containing the course materials and start a Codespace. This Codespace will be owned by the student and persists for 30 days by default, allowing them to revisit the Codespace if the course spans multiple sessions, or to revisit it after the session. Learners are also able to download the materials from their Codespace, allowing them to keep any annotations they make during the course or their solutions to exercises.

By default, a Codespace will open a Visual Studio Code interface in the browser. This makes the interface very portable as it will work on any operating system and is independent of the learner's computer. This greatly reduces the likelihood of problems during setup and means the learner doesn't need to install anything locally on their computer before the session. Visual Studio Code is a powerful and flexible interface, supporting a wide range of programming languages and tools. This includes a built-in Linux terminal allowing command-line commands and other programs to be run in the Codespace.

The Codespace is run within a Docker container, which is defined by the creator of the repository. This allows the educator to define the environment the learner is using, including installing programs in the container, installing packages and dependencies within a programming language environment, or installing Visual Studio Code extensions. This setup will be done automatically when the Codespace begins. This means the educator can make sure all tools are correctly and reliably installed across all learners' environments. The learner will still have to learn to navigate the interface, but there should be many fewer problems relating to setup compared to students locally configuring their own system.

[67] https://xkcd.com/2347/

Comparison of the Main Server or Docker-Based Options

Each cloud-based tool has its pros and cons and quirks. Below you can find a table that summarises them based on our experience.

Name	Type	Pros	Cons
Codespaces	IDE-based	Comprehensive support of a wide range of tasks and extensive ability to set up the environment.	Tie-in to Microsoft ecosystem. Complex configurations can be slow to load.
Posit	IDE-based	Comprehensive R and Python support; seamless GitHub integration	No pre-installed packages result in a long setup time running (and possible crashes), data security concerns, can become expensive for larger teams or more extensive resource needs
PythonAnywhere	IDE-based	Support for Python, hosting a web app and using a bash console; can link teacher and student accounts	No graphics, data security concerns, IPython notebooks are paid-only, teacher and student accounts need to be on the same server (EU and US versions)
Colab	Notebook-based	Easily integrates with Google Drive and GitHub; free GPU access for better performance on larger datasets.	Limited to Python for deep technical tasks, limited amount of pre-installed libraries, data security concerns
My Binder	Notebook-based	Turns a Git repo into a collection of interactive notebooks, free to use	Instances are temporary and can take ages to spin (traffic-dependent and therefore unpredictable), and are mostly Python-oriented
Noteable	Notebook-based	Designed specifically for educational purposes, integrates well with University/VLE login systems, good data security	Pre-installed packages can create issues with additional packages needed, free only for a specific institution makes collaboration harder

Installation Support for Different OS/Devices

Sometimes it may not be possible or desirable to use a cloud-based service. Learners may not have a steady internet connection at home, or may want or need to work on public transport without an internet connection. Online tools often have a limited number of CPU-hours (or a similar restriction) that make them unsuitable for larger projects. Finally, teaching students to run code and tools locally is a valuable skill and one that an educator might want to cultivate. For these reasons, instructors may want to support students running code and tools locally. However, learners may have different operating systems and setups, and this means supporting the installation of software across all machines can be a challenge.

To effectively engage students from the beginning, having additional support for installing tools plays a crucial role. Particularly for courses with a lot of novice students, the tools that will be used in the course should be introduced to students early so they feel comfortable using them. As well as introducing tools in a workshop or lecture, asynchronous support resources (e.g. online documentation or written materials) can help troubleshoot setup issues. These resources should be easy to follow and up to date. This may involve updating them during the first couple of weeks in response to student problems and feedback.

The first workshop of a course could be dedicated to installing software, and making sure the students' computers are set up correctly and that they are comfortable with the tool. There may be device-specific issues, especially for devices other than Windows and Mac laptops such as Chromebooks or tablets. Additionally, students might miss the first week, or new problems may crop up as the course progresses, meaning ongoing support is vital. One-to-one support for individual students and Q&A hours can help make sure students remain supported in terms of their setup throughout the course. Sometimes, overcoming a technical obstacle can be too great a task to

achieve in a session. In these cases, it is possible to suggest that students temporarily use alternative computational platforms (such as lab computers or online platforms) to avoid falling behind during the workshop, and then give extra help after the session to resolve their individual device-related issues.

Supporting Students

When completing a programme of study, students will often need support to overcome technical hurdles. In this section, we discuss how educators might provide this support.

Office Hours and Drop-In Sessions

Learning programming for the first time requires a lot of time and practice. For students to be successful, they will most likely need to practice in their independent study time. However, this will mean that they do not have anyone to ask if they get an error message they don't understand or if they need any guidance on how to progress. If a student cannot progress and has to wait for potentially a week or longer before their next taught session, this can stall progress and demotivate the student.

Office hours or drop-in sessions can be crucial in supporting students between sessions. This could be either individually or in small groups, especially for those who are otherwise struggling to keep on top of the course. Office hours could be scheduled for specific times throughout the week, and drop-in sessions could either be timetabled, arranged using booking software such as Microsoft Bookings or arranged on request by email.

Clear communication to students that this assistance is available and how to reach out if they feel like they need support is important for this to be effective. Doing this consistently from the first session, such as mentioning it once in every class, should help to reach students if they start to struggle at any point in the course. One-to-one or small group sessions can also be really useful for encouraging students and building their confidence, as well as finding out what areas of the course are most commonly misunderstood.

While being available for student support outside of their timetabled teaching can be valuable, it is also important as educators to establish boundaries for our own well-being — for instance, only allowing student meetings to take place during your own working hours and blocking out time for breaks or to focus on other work.

Teams Channels (and Other Group Chat Services)

Providing a managed online chat platform can be effective to allow students a space to discuss the material. The choice of chat service can be strictly limited by University policy and GDPR, despite there being a wide range of alternatives — from commercial, closed-source, hosted versions like Microsoft Teams, Discord, Slack, WhatsApp and so on, through more open-source, locally-hostable services like Mattermost, Zulip, RocketChat, etc. However, it is common to only be allowed to create official chat channels on one of these options, usually Microsoft Teams.

Setting up Teams channels for a course can be effective to allow a space for students to discuss the material, but this requires careful management. Specific channels can guide students to where they can find related materials or post their questions. For instance, an additional code support channel can be created, and students can be informed about how to share their code-related questions specifically. Often, the students who most want to discuss difficulties are also reluctant to admit to issues in a "public space" where teaching staff and other students can observe them. Pseudonymous or anonymous posting in Teams channels hosted by Universities tends to be against policy for safety reasons, but sometimes semi-anonymous posting (where only the teacher can see a poster's identity) can help to relieve social anxiety of students. Tools such as Padlet may be more suited to allowing "low-pressure" anonymous discussion, but anonymous posting requires more moderation to ensure no inappropriate content appears.

Teams, and other equivalent services, provide useful integrations with many of the above services, and can be configured to at least coordinate updates to student material in a centralised way.

AI Assistance

Generative AI tools are rapidly becoming a popular tool for learning to program, or to support learning programming. This has some advantages and disadvantages when it comes to learning programming, but it also has some applications when trying to remove barriers to getting started. Generative AIs have been trained on a wide range of material, including sources like Stack Overflow, where programmers have been asking questions relating to errors and problems with setups for a long time. This means, in theory, generative AI tools should be able to answer at least some questions relating to setup issues.

There are also many ethical issues which may influence your desire to use Generative AI tools in the classroom. Generative AI companies consume large amounts of power in training (and build new, larger, datacentres at high rates, with their own environmental impacts)(IEA 2024). Many generative AI models were trained on copyrighted data without consent (Quintais 2025) and their unmanaged adoption poses significant workers' rights issues. Numerous cases have been recorded of generative AI agents having a negative influence on social relationships. Several papers have also demonstrated that the training sets for most generative AI models used for text generation ("LLMs") have significant Western-USA English-speaking cultural bias, resulting in their outputs being similarly biased (Nyaaba, Wright, and Choi 2024); this results in a "flattening" effect in expression for people interacting with LLMs to "improve" their writing, or when soliciting advice (Agarwal, Naaman, and Vashistha

2024). Given their prominence within discourse, and the fact that people are still using them regardless of the above, it is still relevant to discuss their application to beginning coders.

Sometimes, pasting an error message into a generative AI tool does give useful information that can help to solve a problem. However, there are important limitations for learners. Firstly, generative AI tools can only see the information they are presented with. A tool like ChatGPT can only see the information in its prompts. A tool like GitHub Copilot can be integrated into the development environment and can see, or be provided with, the file structure of the project. More recently, GitHub Copilot can be used in "Agent" mode to interrogate the setup of the system in terms of installing programs, packages or their configuration. This is sporadically successful and can sometimes identify and fix issues, but this is unreliable. Generative AIs cannot generally see the interface the user is looking at, making it difficult to provide information about certain classes of problems.

Another limitation to the information available to the tool comes from the learners themselves. A novice learner may not be able to adequately or accurately describe the problem they're encountering. This can lead to the tool misdiagnosing the problem. Learners with less experience may also struggle to implement reasonable attempts to fix a problem described by a generative AI and not know what follow-up questions to ask. Even if a problem is precisely described, generative AI tools can sometimes fail to suggest the actual fix to a problem. A tutor looking over a student's shoulder in a session may be able to immediately fix the same problem, particularly if they're an expert in the area or have observed an exercise being tackled by lots of students before.

Educators are often also skilled at quickly establishing a learner's level of understanding and providing further help and explanation when necessary. Generative AI tools are not yet good at this as they can often suggest code which uses advanced structures that the learner has not yet encountered, which could overwhelm and confuse the learner, rather than gradually introducing new concepts and structures as the learner progresses.

At their current level of maturity, generative AI tools are unreliable, particularly when it comes to helping novices with limited tacit knowledge of the area being studied. Sometimes they are helpful, and attempting to use one is not a bad idea for a stuck learner, but it often won't work. As generative AI tools improve in their accuracy and become more closely integrated into development environments, their usefulness may improve.

However, a significant issue for generative AI tools is that they can become a tempting crutch. If a student is already underconfident in their skills, the apparent ability of a large language model to simply "solve the problem" can lead to disengagement from the material entirely — a student may substitute reliance on prompting the generative model for solutions for their own critical thought. In the worst cases, which some of us have observed, reliance on generative AI tools can result in the student effectively deskilling as a result. Generally, this pattern follows the form that a student initially develops some small skill in writing code from scratch, but increasingly leans on LLMs to handle "the difficult bits" — over time, increasing amounts of the coding process are handled by the LLM, to the extent that the student is no longer confident in even starting a project without the LLM to provide some initial material to work with.

Conclusions

In this chapter, we have discussed a variety of barriers that students may face when trying to engage with learning activities relating to programming. These are varied, including technical, cultural and practical aspects. There are a number of steps that we, as educators, can take to mitigate these problems and we have outlined and discussed some of the approaches that we have found work for us. A common theme relates to reducing cognitive load — the more we can reduce the level of tacit knowledge, setup and background information, the easier it is to get beginners started. Careful consideration of the order in which skills are taught within a course and across courses can also be helpful for ensuring that students have the correct skills when their studies require it.

Every context is different in terms of the cohort of students, the material being taught and the time and resources available, so there is no "one size fits all" approach. A good technique is to be mindful of and actively identify the barriers students may face, putting yourself in their shoes. From there, it is possible to think of the most suitable mitigations and strategies we can use to reduce the barriers students face and improve their educational experience.

The above collection of advice, rather than aiming to be a definitive guide of dos and don'ts when it comes to removing technology barriers for novice learners, represents a set of wins and fails that the authors have experienced directly in the classroom. Removing technological barriers in programming education is inherently an ongoing process, requiring flexibility, clear communication, and a variety of accessible tools. No single approach will suit every learner, but by combining structured resources, visual guidance, and responsive support, educators can meet students where they are.

Crucially, these useful techniques are most effective when they are embedded

not only within individual courses but also consistently across an entire curriculum. This consistency reduces cognitive load for learners and ensures that valuable skills — such as using GitHub, for example — are reinforced through repeated exposure, making them second nature. As digital platforms and AI assistance continue to evolve, maintaining a careful balance between support and the fostering of independent problem-solving becomes more important than ever. By addressing setup frustrations at both course and programme levels, and prioritising an inclusive environment, educators can lay the essential foundations for students to gain confidence and thrive as they begin their journey in coding.

References

Agarwal, Dhruv, Mor Naaman, and Aditya Vashistha. 2024. “AI Suggestions Homogenize Writing Toward Western Styles and Diminish Cultural Nuances.” In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. Yokohama Japan: ACM. https://doi.org/10.1145/3706598.3713564.

IEA. 2024. “Energy Demand from AI.” https://www.iea.org/reports/energy-and-ai/energy-demand-from-ai.

Nyaaba, Matthew, Alyson Wright, and Gyu Lim Choi. 2024. “Generative AI and Power Imbalances in Global Education: Frameworks for Bias Mitigation.” https://arxiv.org/abs/2406.02966.

Quintais, João Pedro. 2025. “Generative AI, Copyright and the AI Act.” *Computer Law & Security Review* 56: 106107. https://doi.org/10.1016/j.clsr.2025.106107.

Practices To Foster Inclusion and Accessibility in Programming Teaching

Rebecca L. Colquhoun

Department of Earth Science and Engineering, Imperial College London
r.colquhoun23@imperial.ac.uk

Samantha Ahern

Centre for Advanced Research Computing, UCL
s.ahern@ucl.ac.uk

Gule Saman

School of Engineering & Physical Sciences, Heriott-Watt University
s.gule@hw.ac.uk

Brittany Blankinship

Edinburgh Medical School, University of Edinburgh
b.blankinship@ed.ac.uk

Patricia A. Loto

Software Quality Research Group, Universidad Nacional del Nordeste(UNNE)
patricialoto@comunidad.unne.edu.ar

Introduction

We believe that learning programming should be a truly equitable and inclusive process, in which all students are able to take part and succeed. In order to achieve this, we — as programming educators — must continually and critically assess the fundamentals of our teaching techniques. In this chapter, we will first briefly discuss what makes teaching accessible or inaccessible for different groups of students (including but not limited to disabled students and students learning in an additional language), before reviewing different pedagogical approaches often used in teaching programming in light of these principles. These reflections are intended to empower educators to adapt and innovate rather than to act as a prescriptive guidebook.

There are three main aspects to consider when making programming teaching accessible: 1. making the workspace accessible, 2. making the instruction accessible, and 3. considering the implications of the method of delivery. Most universities have a student disability office, who can provide advice on making course materials accessible and will often help students to gain access to assistive technology software. Advice might include making sure to use tagged PDFs or Markdown, or whichever typeface, size and colour combinations are best for use on a presentation. However, these offices may be less familiar with helping students to access programming environments. For some students, simply changing the editor theme will be sufficient. Others may need to experiment with different editors to find one which works with their assistive technology (see e.g. Kramer (2024)). We will not discuss this any further here, but instead refer readers to resources such as Hadwen-Bennett, Sentance, and Morrison (2018).

In this chapter, we focus on the implications of methods of delivery to the accessibility of programming teaching. Along the way, we identify where adaptations to a particular method might make it more suitable or accessible for a certain group of learners. The pedagogical framework that we use (knowingly or unknowingly) to organise our teaching sessions can impact the accessibility of our teaching: we can consider a continuum from transmissive learning (in which knowledge is “transmitted” to students by instructors e.g. through direct instruction) to student-led, discovery based learning (often based on a constructivist model). Programming is a subject which could lend itself well to authentic problem based learning wherein students construct their own knowledge. However, this can present additional barriers to neurodivergent students, who often experience a higher cognitive burden whilst completing discovery-based learning tasks compared to tasks where they were explicitly directed on what to focus on, and compared to their neurotypical peers (Gabay et al. 2025). Problem solving, which is inherent in discovery based learning, places a large burden on working memory (Sweller 1988) which neurodivergent individuals may have weaknesses in (Maehler and Schuchardt 2016). Discovery based learning may also be more suited to expert learners, who have a bank of pre-existing knowledge to work from, than to novices (Kirschner, Sweller, and Clark 2006). This illustrates how teaching using a different pedagogic framework can be accessible for different groups of learners.

Some students may find it hard to generalise, or to engage when examples are not relevant to their chosen domain. Consider gathering feedback on student interests, learning preferences, and backgrounds through a pre-course survey, in order to curate relevant examples and case studies that resonate with student experiences (Kahu, Nelson, and Picton 2017). Engaging students in building course content signals that their opinions shape the course content, fostering a sense of community and ownership. Pre-course surveys also give another opportunity for students to disclose any disabilities or access needs. Some students may not have access to a suitable device either for in class or out

of class activities. Asking about device access in a pre-course survey can help to find solutions (e.g. a loaner laptop or finding a combination of software which works with any assistive technology) before classes begin.

A key aspect of making content accessible is considering cognitive load. In this, the cognitive engagement a task requires can be divided into 'intrinsic load' (the inherent difficulty of the task), 'germane load' (the work put into creating a permanent store of knowledge) and 'extraneous load' (the way information is presented) (Sweller 1988; Paas, Renkl, and Sweller 2003). Everyone has a limit to their working memory, and when the total cognitive load exceeds this, learning is impacted. The germane and extraneous load of a task can vary between learners (for example due to neurodivergence (Le Cunff, Giampietro, and Dommett 2024) or due to learning in an additional language) and is also influenced by how material is presented. In the chapter, we discuss some ways that educators can manage the extraneous load in their classrooms, to ensure that students have capacity to engage in the germane load of building long-term mental schemas of programming logic (Sweller 1994).

Generally, designing teaching sessions and materials around Universal Design for Learning (UDL) principles helps maximise accessibility. UDL centers on students having agency and designing materials to have multiple options for representation, engagement and action (CAST 2024).

Ordering of Material

In teaching programming there are different approaches to how material is presented. For example, Selby identifies four approaches: 1. code analysis, 2. building blocks, 3. simple units and 4. full systems (Selby 2011). Code analysis begins with students analysing and understanding existing code before going onto writing pseudocode. Only later do students write full programs. This approach provides a structured way for students to first focus on programming logic before having to worry about syntax, which may reduce cognitive load. However some students may struggle with knowledge transfer between writing pseudocode and writing compilable code (Glaser, Hartel, and Garratt 2000).

In other approaches (building blocks and simple units), students learn small units of code (individual constructs or set coding "phrases" respectively) and gradually build up to writing longer programs. These can lead to fragmented understanding, where students are not sure how to combine the components introduced together, and the early focus on syntax may be overwhelming for some (Selby 2011). However, it does allow students to build up a toolbox of reusable code, which they can then apply in other scenarios, reducing the amount of recall and typing of precise syntax which is required which may pose a challenge for some students. The final approach that Selby (2011) outlines is full systems, where students are immersed in a real-life problem and programming concepts are introduced as needed. This is close to the discovery-led learning discussed previously.

Classroom Models

These different approaches to teaching programming can be delivered in different ways. We have identified six different classroom models, though there is significant overlap and they are not mutually exclusive. We will now discuss each approach in turn, identifying what makes them accessible, and where they may present an accessibility barrier. Where possible, we identify possible modifications to the model to increase accessibility.

Participatory Live Coding

Live coding involves an instructor writing code live at the front of the classroom, whilst narrating what they are doing. It becomes participatory by students following along, writing and executing code at the same time (Nederbragt et al. 2020). Participatory live coding has many advantages, hence why it forms a key tenet of the Carpentries approach to teaching coding (The Carpentries 2025). For many students, watching someone code and following along is far more engaging than watching someone present a slide deck. In particular, it allows students the chance to ask questions and for the instructor to respond and adapt the teaching accordingly.

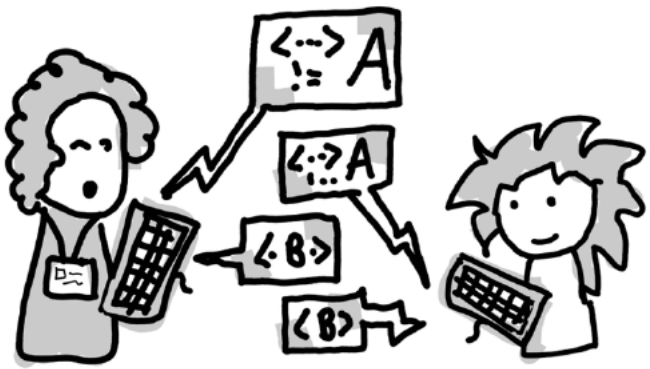


Figure 13.1: I code it. Then you code it. Repeat.

The instructor needing to type out code also reduces the pace of instruction: it is all too easy to quickly flash up some code on a slide, and have moved onto the next slide before the student has been able to write it down, which is a problem if slide decks are not provided in advance. Equally, it helps keep students on track. The lesson progresses at a set pace reducing the opportunity for students to take exit ramps or get lost on an unrelated tangent. However, the set pace can be an accessibility barrier for some, as it still depends on the instructor being able to judge an appropriate pace that will not leave students behind. Some students may require a slower pace

than others, for example dyslexic students may struggle to accurately copy down material from the screen and so will need more time to do this (Blampain, Gosse, and Van Reybroeck 2021).

As well as students requiring different amounts of time to copy down information from the board, some may find this difficult or impossible to do. The participatory live coding approach is particularly challenging for students who use a screen-reader, since they would have to balance auditory input from their computer with the instructor talking. To modify the approach for students who cannot see the board or accurately transcribe material, the instructor could provide code snippets or help people follow along using a git repository and tags (Bostian 2019; Sutton 2018). A collaborative notes document may be another approach. These ideas reduce the need for students to copy from the board, instead following along with small code chunks at a time. Students are also aided by the instructor making sure they verbally describe the actions and perhaps even dictating each word they type. This helps students follow along without needing to look up for each word typed.

In a participatory live coding approach, students also benefit from watching how the instructor codes: they are able to model the approach to a problem and the making and solving of mistakes. Explicitly modelling problem solving is particularly helpful for neurodivergent or anxious students for whom arriving at an error, mistake or block may become a barrier for learning rather than an opportunity.

Some students may benefit from switching to a flipped classroom approach where the instructor is pre-recorded doing a small chunk of coding (see *A Video Introduction to Live Coding Part 2* (2021) for an example of a single chunk), and students can follow along. This could either be done in the classroom, with instructors on hand to help students as needed, or be done at home and the classroom time be dedicated to longer problems. This approach also allows for students to turn on captions if needed.

Alternatively, the session could be live streamed, so students can see the projected screen more easily on their own computers, adjust the volume and can turn on captions if they wish, whilst still being able to ask questions.

Pair Programming

Instructors may want to include time where students independently solve programming problems, either as part of a broader lesson, or as part of inquiry based learning. One way to do this is through pair-programming, where students work in pairs to solve a problem, each taking turns to be the 'driver' and the 'navigator'. This concept is discussed in more detail in another chapter in this book (Desvages et al. 2026). Pair programming has been found to lead to better outcomes in introductory programming courses compared to courses where students program alone (Hanks et al. 2011; Faja 2014).

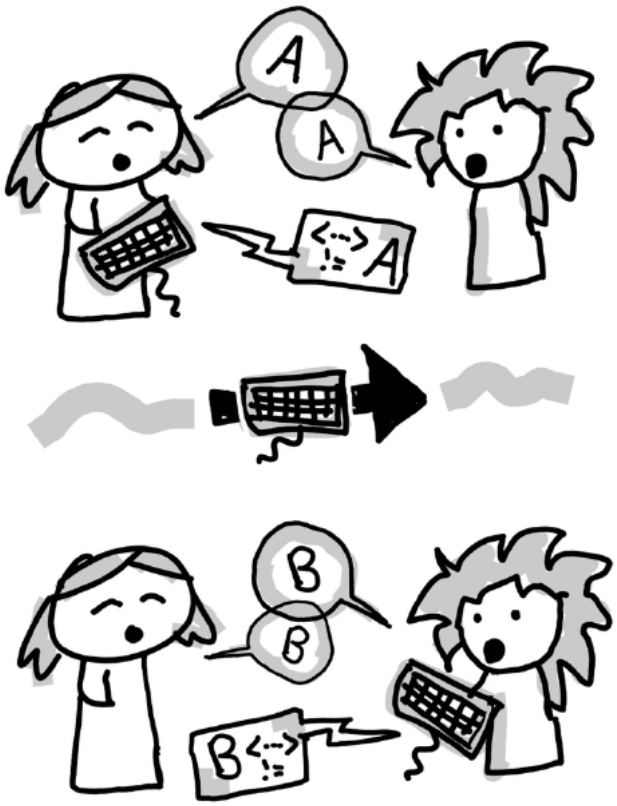


Figure 13.2: I drive, you navigate. Now let's switch who's driving.

A key advantage of pair programming is that it allows students to see different approaches to solving the same problem. For weaker or less confident students, this can help them build confidence and can model how to break down a problem. However, for others, students may compare themselves negatively to one another. One way to alleviate this is to switch partners around, which allows students to see different strengths and weaknesses in coding. For students who struggle to keep focussed, pair programming can be helpful, since both students are required to maintain active engagement with the task, frequently switching roles. It can also help keep students on task and working at the right pace, if the pairs are chosen carefully.

Pair programming can also serve to reduce the cognitive burden on students (Tsai, Yang, and Chang 2015) by allowing students to distribute tasks of high and low cognitive burden amongst themselves (Sands 2019; Winkler and Flatscher 2023). In classrooms where students have varying levels of experience with programming, pairs could consist of an expert and a novice where the expert (who would ordinarily experience lower extraneous load due to familiarity e.g. with the programming environment) can take on more of the extraneous tasks, allowing the novice to have more capacity for germane cognitive load.

A 2023 study from the University of Toronto Scarborough (Ali and Joordens 2023) found that there has been an increase in social anxiety amongst both higher education students and adolescents since the COVID-19 pandemic. Socially anxious students can experience anticipatory anxiety, impacting their abilities to communicate effectively with others and negatively impacting on their cognitive and emotional engagement. This could be reduced through the use of small-group icebreakers in sessions

prior to the introduction of pair programming. These activities help build a sense of familiarity and community within the learning environment, potentially reducing levels of anxiety (Archbell and Coplan 2022).

Whilst pair programming can reduce some of the cognitive load of programming, it can also introduce new sources of extraneous cognitive load since students now need to manage the social interaction associated with group work. It is therefore important to scaffold the task and provide a framework for how students should give feedback or interact during the exercise. This could look like modelling pair programming as part of the 'lecture' or by clearly explaining what will happen during the task. It should also include clearly signposting when switches in role and task will happen. Students may also try to impose one way of solving a problem as the 'correct' way, making it difficult for (neurodivergent) students who think differently, and expectations around this should be set during the task introduction.

Pair programming is often thought of as two people using one computer, switching over who is physically typing on the keyboard. However, students may find it difficult to use a computer with someone else's accessibility settings. It can therefore be helpful to use a shared workspace but on separate computers, so long as it is clear who is 'driving' at any given time. Students may need to balance working on the screen with auditory input which may be difficult for D/deaf or hard of hearing students or those who use a screen-reader. Some students may need their pair to work in a separate room to reduce background noise, which may be easier if pair-programming is being done virtually (e.g. through the use of breakout rooms).

Explicitly Modelling Problem Solving

When learning to program, whether in a classroom setting or independently, it is essential to recognise that programming is fundamentally about problem-solving and critical thinking as discussed by Govender (Govender 2007). The syntax of a programming language varies between languages, but what truly matters is developing the ability to think systematically about a problem and design an effective solution. Explicit modelling of the different stages in problem solving can be helpful for many students.



Figure 13.3: These are the steps I would take to solve this problem.

A helpful analogy is preparing a cup of tea. There are countless variations: some people prefer milk, others like sugar; some want herbal, others like it strong. The method you use depends on the preferences of the person you are making it for and any constraints involved (available ingredients, desired strength, etc.). You begin by understanding the requirements, clarifying constraints, and considering expectations, then decide on the steps to achieve the desired result. Programming is similar: syntax is just the final step in communicating a solution, but the real issue is clear thinking and structured planning.

The process of problem solving can be broken down into several steps. Some students will automatically do this, whilst others (particularly those who struggle with executive dysfunction) will benefit from modelling of the different stages and being provided with a checklist to follow:

- Understanding the problem
- Identifying constraints
- Identifying the expected inputs
- Considering the target audience or users
- Clarifying expectations
- Knowing about the desired outcomes (outputs)
- Designing the solution
- Testing possible solutions
- Troubleshooting and refining the approach

By foregrounding problem-solving instead of syntax, students can focus on creative and strategic thinking, which lowers barriers for those who might lack confidence with coding rules. However, not all learners will find abstract planning easy; some thrive by diving straight into code. For these learners, scaffolding the design stage can be helpful. Progressing from checklist to pseudocode, flowchart, and code supports learners with diverse preferences and reinforces both problem solving and implementation skills.

Accessible and inclusive programming education goes far beyond teaching language syntax. Focusing on a structured problem-solving process supports learners in developing analytical thinking, creativity, and confidence. By integrating scaffolding tools such as flowcharts, pseudocode, and concrete examples, educators can create a classroom environment where all students feel equipped to participate and succeed. Encouraging learners to think critically about problems, consider target audience and constraints, and work systematically before coding is important.

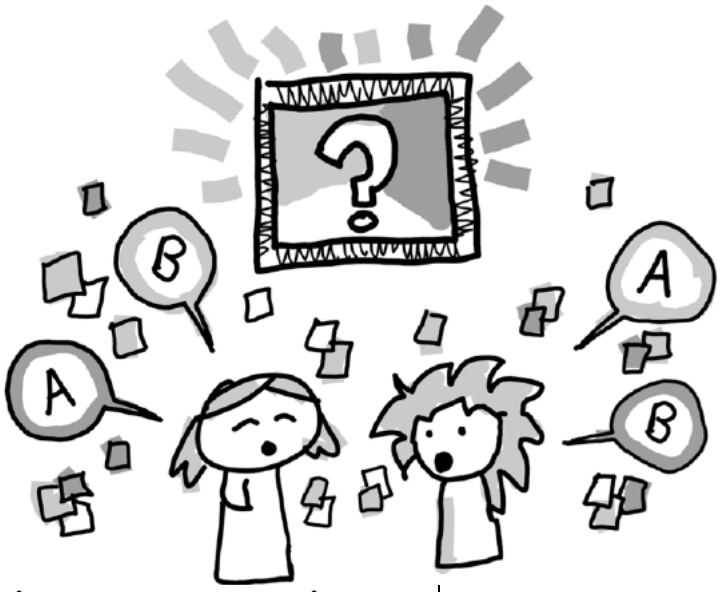


Figure 13.4: We learn new things when needed as part of a bigger project.

Discovery Based Learning

As we discussed in the introduction, instructors may prefer to use a discovery-based learning approach rather than the more direct-instruction that could lend itself well to participatory live coding. We previously discussed the cognitive load inherent in discovery based learning, but here consider other aspects of accessibility for this method of teaching.

The advantage of discovery based learning is that students have a reason for everything they learn. When students arrive at a problem, or need a new tool, they discover what is most suitable, and this can be more motivating to some students. However, some students may be overwhelmed by the scale of the task and struggle to break down a problem into manageable chunks. Part of the solution to this is making sure students are comfortable asking questions, rather than feeling like they have been 'abandoned' to teach themselves. This can be aided by having enough instructors and offering less obtrusive ways to ask questions, for example using a webchat or post-it note flag system. It may also be helpful to combine discovery based learning with some modelling of problem solving in a larger group. Another approach is to combine discovery based learning with pair-programming, with the inherent advantages and disadvantages we previously discussed. Finally, some students may struggle with the generalisation required and transfer to long-term memory: they may successfully identify a tool as suitable for this problem, but when it comes to the next problem they may start from scratch rather than being able to apply their existing knowledge. Part of this is developed with repeated exposure to different problems, but prompts from instructors may also help students realise how ideas can be generalised.

'Traditional' Lecture and Lab Sessions

The 'traditional' model of a lecture with a lab/practical session (e.g. Prather and Schlesinger 1978) combines instructor-led teaching with an opportunity to partake in practice of the material and/or discovery based learning. However, in the lecture session concepts or ideas may be introduced at a pace where some students become anxious about trying to process large amounts of content in real-time. This can be exacerbated by the use of dismissive language such as "just" or "even a monkey can do...". In addition, neurodivergent students can struggle to retain focus when there is a lot of information or lack of interaction.

The accessibility of lecture sessions can be improved by providing the session materials in advance. This enables students to engage with them at their own pace and identify any questions they may have for the lecture session. In addition, the inclusion of interactive components in the lecture session such as anonymous quizzes or discussions, enables reinforcement of learning without potentially causing discomfort amongst neurodivergent learners. However, it is important to be mindful that too much interaction may lead to cognitive overload. The regular use of lecture capture can make recordings available to students to review content as and when needed, reducing the risk of anxiety from a fear of not being able to keep up.

Figure 13.5: I talk, you listen. Then you do, I help.



When done well, the combination of lecture and lab sessions can help facilitate the navigation of the liminal space between a threshold concept being introduced and it being understood. Our learners, may not navigate this linearly, but will go through the stages of developing:

- an abstract (or theoretical) understanding of a concept;
- a concrete (or applied) understanding — the ability to write a computer program, for instance;
- the ability to go from an understanding of the abstract concept to software design or concrete implementation;
- an understanding of the rationale for learning and using the concept; and
- an understanding of how to apply the concept to new problems — problems beyond those given as homework or lab exercises (McCartney et al. 2009).

Knuth (1991) spoke about the close relationship between theory and practice as follows:

> "The main point I want to emphasize [...] is that both theory and practice can and should be present simultaneously. Theory and practice are not mutually exclusive; they are intimately connected. They live together and support each other." (Knuth 1991, 3)

The interaction between theory and practice can be promoted through careful design of lab instructions so they highlight key concepts and links that students should explore during the lab.

Some students may struggle with moving from theory to practice. They may make a sequence of unsuccessful attempts to implement, and iterate on, a solution, or get stuck in theory-oriented discussions without trying any of their ideas in practice. To alleviate this risk and support connections between theory and practice, an educator could intersperse references to manual and guidance materials within a task or problem sheet, ensuring relavance and ease of use for novice learners in particular. Instructors can also include reminders that students should save, compile and run their code regularaly, even if it is not yet working as intended (Eckerdal, Berglund, and Thuné 2024).

Flipped Classroom

An alternative approach to structuring learning is the flipped classroom model, wherein students engage with learning materials before class, and classroom time is used to engage in problem solving activities (Bergmann and Sams 2012; Advance HE, n.d.). By providing learning materials in advance (most commonly through pre-recorded videos, although this may also be in the form of text or audio), students can engage at their own pace. For some students, removing any time pressure enables them to better engage with the material, since they can pause to take notes, take a break or consult other materials during the course of the 'lecture'. However, effective use of the flipped classroom requires students to engage with the material before class (Moraros et al. 2015; Gross et al. 2015; Forsey, Low, and Glance 2013) and some students may struggle with the lack of direction, either not engaging with the material or becoming distracted by tangents. In a flipped classroom model, it is particularly important to ensure that the structure and signposting of materials is clear, as this can easily become a source of confusion for students.

Having time between engaging with the flipped classroom materials and the in-person session enables students to identify any questions they have in advance, and it is important that materials are released well in advance of synchronous sessions to allow for meaningful engagement. In order to further ensure that students feel comfortable to ask questions, an anonymous question box could be introduced, which is used to guide the in-person sessions. However, if a student misunderstands something early on in the asynchronous teaching, this may lead them to misunderstand the rest of the topic, whereas they may have normally been able to clarify their understanding more immediately.

Students can engage with the asynchronous portion in a location they feel most comfortable. For some students with disabilities, traditional lecture theaters may not be accessible, for example because of the fluorescent lighting or seating options. However, whilst the lecture portion may be more accessible using a flipped classroom model, the practical portion may be less accessible, as they are often centered on working as a group. Group activities have been shown to not necessarily result in long-term learning for neurodivergent students, and they may prove overwhelming or impose an additional cognitive burden on some students (Durgungoz and Durgungoz

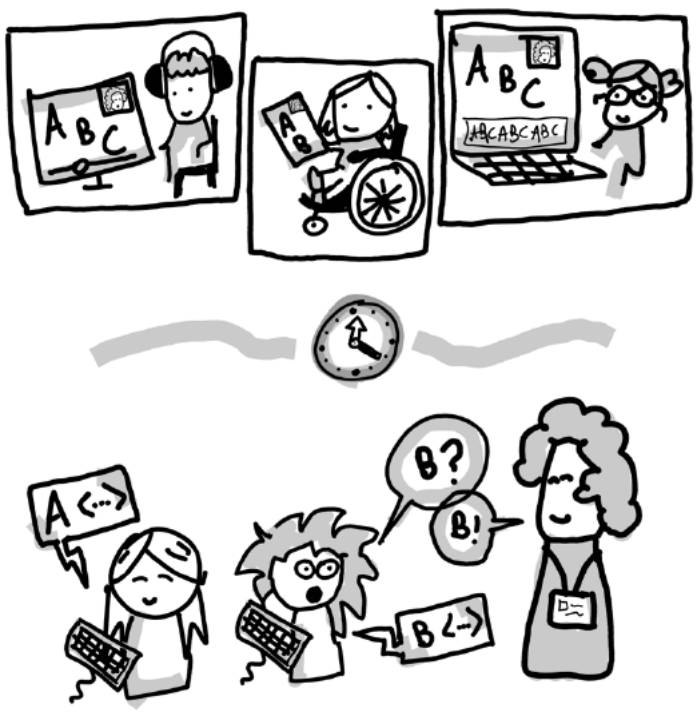


Figure 13.6: You watch and read at home. Later, we do and discuss in class.

2025). Some students, particularly those who are less confident in the course materials may also feel unprepared to engage in group discussions, if they have not fully understood the material for example.

One way to support students in group activities is to center discussions around artefacts that students have prepared beforehand such as a solution to a coding challenge (Dusek et al. 2018). This helps students prepare for likely topics of discussion beforehand, and could be a motivator for some students to engage more diligently in the asynchronous work. Even if it is not desirable for students to complete all the practical work beforehand, it can be advantageous to provide practice exercises to help students prepare for the interactive session to both help students prepare and let them know what to expect.

Framing either in-class or preparatory practical exercises as Parson's problems or as part of a series of 'fading' activities may also be beneficial. In some settings, students could have the opportunity to engage in the development of an artefact via coding challenges with the in-person session becomes the lab session, as described above. If these are designed well, they may incorporate components of pair programming, and stimulate deep learning through the integration of theory and practice with the support of the module staff. Providing material online and in advance provides an opportunity to ensure materials are accessible. For example, it is important to make sure that any recorded lectures or demonstrations have captions available, and that there are image descriptions and audio description of what is displayed on screen. Resources such as the STEMM baseline accessibility guidance are a good starting place (Tyson 2022).

Conclusion

There is no singular approach to teaching programming that will always be appropriate, regardless of whether you consider this from a pedagogical or accessibility standpoint. Here we have outlined six different approaches to teaching programming, and what attributes make them accessible, or not, to different groups of learners. The ideas presented are not meant to be exhaustive or prescriptive; indeed, some of the adaptations that we suggest for one method, may be precisely what you need to make a session accessible for a student even if you have made different pedagogical design choices. By outlining a range of approaches, we hope that this chapter will give educators the starting blocks they need to design sessions that give all their learners the best chance of success.

References

A Video Introduction to Live Coding Part 2. 2021. https://www.youtube.com/watch?v=SkPmwe_WjeY.

Advance HE. n.d. "Flipped Learning." https://www.advance-he.ac.uk/knowledge-hub/flipped-learning-0.

Ali, Hannah, and Steve Joordens. 2023. "Social Anxiety in University Students: Towards an Intentional Life-Skills Based Prevention Model." *Journal of Mental Health & Clinical Psychology* 7 (3): 19–36. https://doi.org/10.29245/2578-2959/2023/3.1282.

Archbell, Kristen A., and Robert J. Coplan. 2022. "Too Anxious to Talk: Social Anxiety, Academic Communication, and Students' Experiences in Higher Education." *Journal of Emotional and Behavioral Disorders* 30 (4): 273–86. https://doi.org/10.1177/10634266211060079.

Bergmann, Jonathan, and Aaron Sams. 2012. *Flip Your Classroom: Reach Every Student in Every Class Every Day*. 1st ed. Eugene, Or Alexandria, Va: International Society for Technology in Education.

Blampain, Elise, Claire Gosse, and Marie Van Reybroeck. 2021. "Copying Skills in Children with and Without Dyslexia." *Reading and Writing* 34 (4): 859–85. https://doi.org/10.1007/s11145-020-10094-6.

Bostian, Emma. 2019. "Using Git Tags to Version Coding Tutorials. Dev.to." March 16, 2019. https://dev.to/emmabostian/using-git-tags-to-version-coding-tutorials-39cc.

CAST. 2024. "Universal Design for Learning Guidelines Version 3.0. CAST UDL Guidelines." 2024. https://udlguidelines.cast.org.

Desvages, Charlotte, Brittany Blankinship, Umberto Noè, and Pawel Orzechowski. 2026. "Structured Group Work with Assigned Asymmetrical Roles and Switching: Lessons from Pair Programming Across Disciplines." In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Durgungoz, Fatma Canan, and Ahmet Durgungoz. 2025. "'Interactive Lessons Are Great, but Too Much Is Too Much': Hearing Out Neurodivergent Students, Universal Design for Learning and the Case for Integrating More Anonymous Technology in Higher Education." *Higher Education*, January. https://doi.org/10.1007/s10734-024-01389-6.

Dusek, Jeff, Daniela Faas, Emily Ferrier, Robyn Goodner, Alisha Sarang-Sieminski, Adva Waranyuwat, and Alison Wood. 2018. "Proactive Inclusion of Neurodiverse Learning Styles in Project-Based Learning: A Call for Action." In *2018 ASEE Annual Conference & Exposition Proceedings*, 30891. Salt Lake City, Utah: ASEE Conferences. https://doi.org/10.18260/1-2--30891.

Eckerdal, Anna, Anders Berglund, and Michael Thuné. 2024. "Learning

Programming Practice and Programming Theory in the Computer Laboratory." *European Journal of Engineering Education* 49 (2): 330–47. https://doi.org/10.1080/03043797.2023.2294953.

Faja, Silvana. 2014. "Evaluating Effectiveness of Pair Programming as a Teaching Tool in Programming Courses." *Information Systems Education Journal* 12 (6): 36–44.

Forsey, Martin, Mitchell Low, and David Glance. 2013. "Flipping the Sociology Classroom: Towards a Practice of Online Pedagogy." *Journal of Sociology* 49 (4): 471–85. https://doi.org/10.1177/1440783313504059.

Gabay, Yafit, Lana Jacob, Atil Mansour, and Uri Hertz. 2025. "Computational Markers Show Specific Deficits for Dyslexia and ADHD in Complex Learning Settings." *Npj Science of Learning* 10 (1): 38. https://doi.org/10.1038/s41539-025-00323-4.

Glaser, Hugh, Pieter H. Hartel, and Paul W. Garratt. 2000. "Programming by Numbers: A Programming Method for Novices." *The Computer Journal* 43 (4): 252–65. https://doi.org/10.1093/comjnl/43.4.252.

Govender, I. 2007. "Experiences of Learning and Teaching: Problem Solving in Computer Programming." *African Journal of Research in Mathematics, Science and Technology Education* 11 (2): 39–50. https://doi.org/10.1080/10288457.2007.10740620.

Gross, David, Evava S. Pietri, Gordon Anderson, Karin Moyano-Camihort, and Mark J. Graham. 2015. "Increased Preclass Preparation Underlies Student Outcome Improvement in the Flipped Classroom." Edited by Mary Lee Ledbetter. *CBE–Life Sciences Education* 14 (4): ar36. https://doi.org/10.1187/cbe.15-02-0040.

Hadwen-Bennett, Alex, Sue Sentance, and Cecily Morrison. 2018. "Making Programming Accessible to Learners with Visual Impairments: A Literature Review." *International Journal of Computer Science Education in Schools* 2 (2): 3–13. https://doi.org/10.21585/ijcses.v2i2.25.

Hanks, Brian, Sue Fitzgerald, Renée McCauley, Laurie Murphy, and Carol Zander. 2011. "Pair Programming in Education: A Literature Review." *Computer Science Education* 21 (2): 135–73. https://doi.org/10.1080/08993408.2011.579808.

Kahu, Ella, Karen Nelson, and Catherine Picton. 2017. "Student Interest as a Key Driver of Engagement for First Year Students." *Student Success* 8 (2): 55–66. https://doi.org/10.5204/SSJ.V8I2.379.

Kirschner, Paul A., John Sweller, and Richard E. Clark. 2006. "Why Minimal Guidance During Instruction Does Not Work: An Analysis of the Failure of Constructivist, Discovery, Problem-Based, Experiential, and Inquiry-Based Teaching." *Educational Psychologist* 41 (2): 75–86. https://doi.org/10.1207/s15326985ep4102_1.

Knuth, Donald E. 1991. "Theory and Practice." *Theoretical Computer Science* 90 (1): 1–15. https://doi.org/10.1016/0304-3975(91)90295-D.

Kramer, Nimrod. 2024. "10 Best Code Editors with Accessibility Features. Daily.dev." May 16, 2024. https://daily.dev/blog/10-best-code-editors-with-accessibility-features.

Le Cunff, Anne-Laure, Vincent Giampietro, and Eleanor Dommett. 2024. "Neurodiversity Positively Predicts Perceived Extraneous Load in Online Learning: A Quantitative Research Study." *Education Sciences* 14 (5): 516. https://doi.org/10.3390/educsci14050516.

Maehler, Claudia, and Kirsten Schuchardt. 2016. "Working Memory in Children with Specific Learning Disorders and/or Attention Deficits." *Learning and Individual Differences* 49 (July): 341–47. https://doi.org/10.1016/j.lindif.2016.05.007.

McCartney, Robert, Jonas Boustedt, Anna Eckerdal, Jan Erik Moström, Kate Sanders, Lynda Thomas, and Carol Zander. 2009. "Liminal Spaces and Learning Computing." *European Journal of Engineering Education* 34 (4): 383–91. https://doi.org/10.1080/03043790902989580.

Moraros, John, Adiba Islam, Stan Yu, Ryan Banow, and Barbara Schindelka. 2015. "Flipping for Success: Evaluating the Effectiveness of a Novel Teaching Approach in a Graduate Level Setting." *BMC Medical Education* 15 (1): 27. https://doi.org/10.1186/s12909-015-0317-2.

Nederbragt, Alexander, Rayna Michelle Harris, Alison Presmanes Hill, and Greg Wilson. 2020. "Ten Quick Tips for Teaching with Participatory Live Coding." Edited by Francis Ouellette. *PLOS Computational Biology* 16 (9): e1008090. https://doi.org/10.1371/journal.pcbi.1008090.

Paas, Fred, Alexander Renkl, and John Sweller. 2003. "Cognitive Load Theory and Instructional Design: Recent Developments." *Educational Psychologist* 38 (1): 1–4. https://doi.org/10.1207/S15326985EP3801_1.

Prather, Ronald E., and Judith D. Schlesinger. 1978. "A Lecture/Laboratory Approach to the First Course in Programming." *ACM SIGCSE Bulletin* 10 (1): 115–18. https://doi.org/10.1145/990654.990599.

Sands, Philip. 2019. "Addressing Cognitive Load in the Computer Science Classroom." *ACM Inroads* 10 (1): 44–51. https://doi.org/10.1145/3210577.

Selby, Cynthia. 2011. "Four Approaches to Teaching Programming." In.

Sutton, Marcy. 2018. "Live Coding Accessibility. Marcysutton.com." June 28, 2018. https://marcysutton.com/live-coding-accessibility.

Sweller, John. 1988. "Cognitive Load During Problem Solving: Effects on Learning." *Cognitive Science* 12 (2): 257–85.

https://doi.org/10.1207/s15516709cog1202_4.

———. 1994. “Cognitive Load Theory, Learning Difficulty, and Instructional Design.” *Learning and Instruction* 4 (4): 295–312. https://doi.org/10.1016/0959-4752(94)90003-5.

The Carpentries. 2025. “Live Coding Is a Skill. Instructor Training.” June 4, 2025. https://carpentries.github.io/instructor-training/17-live.html.

Tsai, Chia-Yin, Ya-Fei Yang, and Chih-Kai Chang. 2015. “Cognitive Load Comparison of Traditional and Distributed Pair Programming on Visual Programming Language.” In *2015 International Conference of Educational Innovation Through Technology (EITT)*, 143–46. Wuhan, China: IEEE. https://doi.org/10.1109/EITT.2015.37.

Tyson, Jim. 2022. “Accessible STEMM Introduction. Make Things Accessible.” October 21, 2022. https://www.makethingsaccessible.com/guides/accessible-stemm-introduction/.

Winkler, Till, and Rony G Flatscher. 2023. “Cognitive Load in Programming Education: Easing the Burden on Beginners with REXX.” In *Central European Conference on Information and Intelligent Systems*, 171–78. Faculty of Organization; Informatics Varazdin.

Overcoming Coding Anxiety: Lowering the Stakes and Making It Fun

Tiago A. Marques

Departamento de Biologia Animal, Faculdade de Ciências da Universidade de Lisboa; Centre for Research into Ecological and Environmental Modelling, University of St Andrews
tiago.marques@st-andrews.ac.uk

William P. Kay

School of Biosciences, Cardiff University
kayw@cardiff.ac.uk

Christopher Aldous Oldnall

School of Mathematics and Maxwell Institute for the Mathematical Sciences, University of Edinburgh; Edinburgh Medical School, University of Edinburgh
chris.oldnall@ed.ac.uk

Chris Sutherland

Centre for Research into Ecological and Environmental Modelling, University of St Andrews
css6@st-andrews.ac.uk

Robert S. Young

Edinburgh Medical School, University of Edinburgh; Zhejiang University - University of Edinburgh (ZJE) Institute
robert.young@ed.ac.uk

Coding is increasingly acknowledged as a core component of data analysis across disciplines, yet it is often presented to students as a tool rather than a primary focus of study. For these students, it is likely that they have no prior exposure to coding. These initial interactions will define a long-term relationship with programming, and as a consequence, how they navigate an increasingly data-centric world. Understanding and addressing barriers to learning coding is essential to support learning, and here we consider one major barrier: coding anxiety.

Coding anxiety, or programming anxiety is defined as 'a psychological state engendered when a student experiences or expects to lose self-esteem in confronting a computing situation' (Connolly, Murphy, and Moore 2009). It may be driven, in part, by other anxieties, such as computer anxiety (Chua, Chen, and Wong 1999; Meinhardt-Injac and Skowronek 2022) or statistics anxiety (Zeidner 1991), among others. While one may assume that "digital natives" (Bennett, Maton, and Kervin 2008) experience a lower level of computer anxiety, this assumption may not naturally translate to coding anxiety (Nolan and Bergin 2016). Factors such as variable exposure to coding, an over-reliance on connected digital ecosystems during childhood (*How's Life for Children in the Digital Age?* 2025), or disciplinary background, may influence levels of anxiety. For example, different levels of coding anxiety exist for mathematics and biomedical undergraduate students (Miller and Pyper 2023), with the former finding coding to be more enjoyable, and hence having lower anxiety. More broadly, student identity and background can also contribute to anxiety-linked barriers to learning, such as gender (Forrester et al. 2022) or learning in a non-native language (Kaur and Newell 2024).

Given the multifaceted nature of coding anxiety (PAN and Harun 2025), here we draw on our collective experience of teaching coding, provide perspectives on coding anxiety, and offer strategies to mitigate it. We focus specifically on students for whom coding is not their core academic focus, but rather a necessary skill set required when taking classes involving statistics, data analysis, experimental design, and modelling.

Recognising and Refocusing Anxiety

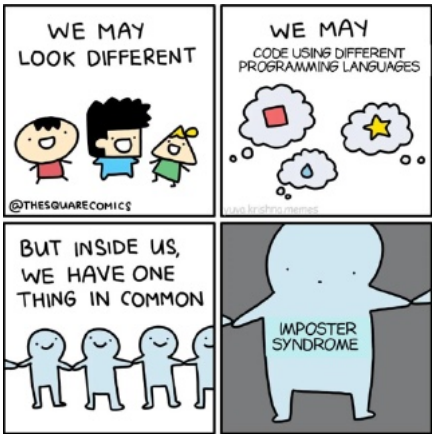
Anxiety sustains itself through a self-perpetuating cycle (Mkrtchian et al. 2017). It is believed to start with a trigger (e.g., a bad experience), which leads to future avoidance of associated triggers in an attempt to feel some short-term relief (Hofmann and Hay 2018). However, avoidance can lead to heightened anxiety in the long term (Mkrtchian et al. 2017). Since the brain learns that avoidance decreases anxiety, even if only temporarily, it may eventually prevent engagement with the trigger entirely. If the trigger is computer code, then coding becomes something to avoid, and hence coding anxiety is born. In contrast, confronting the trigger can build confidence and reduce anxiety over time (Kampmann, Emmelkamp, and Morina 2018; Craske et al. 2014).

We believe that recognising coding anxieties upfront is an important first step towards demystifying it in the classroom. Once students realise that this is an identified problem, common to many rather than something affecting them alone, they are one step closer to overcoming it. Humour, including self-deprecating jokes, describing how, despite now being the teacher, you once suffered from coding anxiety, can go a long way towards making students feel better[68]. Cartoons may also be an effective way of making anxiety itself less daunting. If existing online content is not enough, and making cartoons is not one's gift, AI-generated images could easily be

[68] Even if you've never experienced coding anxiety - lucky you; most of us have, at least to some extent - do take one for the team, and say you have too.

created for that purpose - just make sure people have 5 fingers!

Figure 14.1: An example of a humorous cartoon that normalises coding anxiety. (https://www.thesquarecomics.com/)



Given anxiety's negative feedback loop, the best way to combat anxiety is to break this vicious cycle (Figure 14.2). As an instructor or teacher, this can be achieved by providing positive experiences with coding[69]. Importantly, after acknowledging that coding anxiety is real, the solutions to it should be presented along with how overcoming their anxiety will be supported, e.g., reflecting on personal experience. Offering your experiences and providing a very visible role could serve as a powerful catalyst for students to realise they can overcome their own coding anxiety too.

Gauging a class's perception towards coding early should prove useful in lowering overall anxiety. Collecting information about *a priori* fears or causes of anxiety, including - but not exclusively - coding anxiety, as well as discussions about course expectations, provides clues to what might be good strategies to adopt. In-class feedback tools like Mentimenter[70], allow students to provide feedback anonymously and in real time to make the course more accessible in an informal, friendly, and interactive way. This helps generate useful ideas, including specific suggestions, for how to reduce anxiety. Additionally, having students reflect on how anxious they are about coding might allow them to identify shared problems and anxieties across the class; a problem shared is a problem halved.

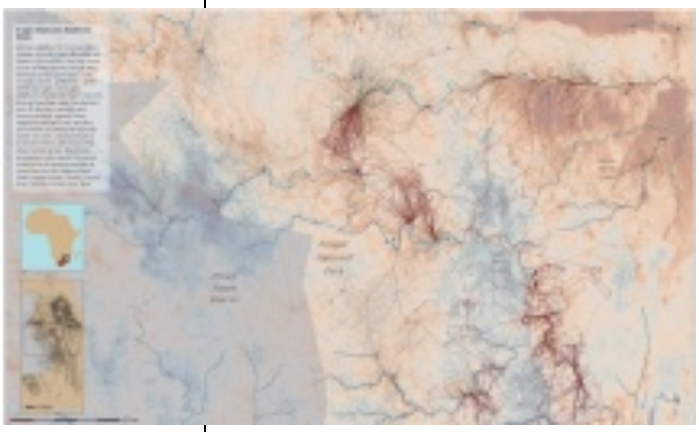
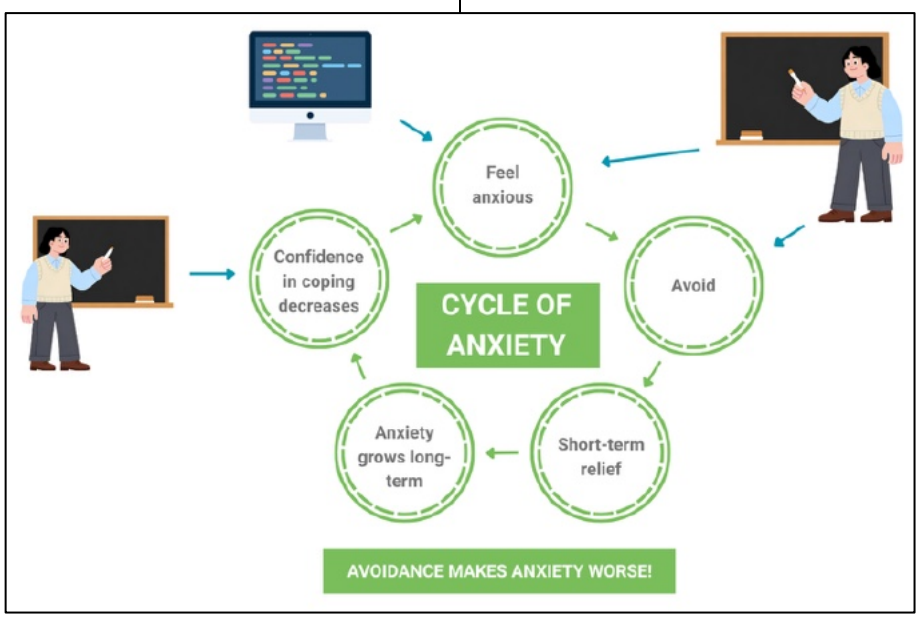


Figure 14.3: Example of a beautiful visualisation created using R (a map of elephant movements through Kruger National Park). Produced by Dr Pratik Gupte for the British Ecological Society Movement Ecology Special Interest Group's MoveMap competition (2021), and reused here with his permission.

Figure 14.2: The cycle of anxiety. Coding is identified as the trigger and the teacher serves as a diffuser to intervene in multiple places and in multiple ways to break the cycle. Adapted from https://www.mentalyc.com/



[69] For some of us, ease with coding developed through repeated practice and the satisfaction of automating otherwise repetitive tasks. For others, it emerged more gradually and holistically, as their overall perception of coding improved over time through continued engagement.

[70] http://www.mentimenter.com

Building Confidence Through Active Learning

Early hands-on activities focusing on the end product of coding, rather than the process of getting there has the benefit of demonstrating value without the need to actually code in the first instance. Even exercises where students simply copy-paste code, without knowing why it does what it does (but being told that later they will be able to code it themselves), could go a long way in reducing anxiety. Such exercises allow students to see how useful coding is for tasks they would like to do, but which would be impossible manually. Focusing on the outcomes students are interested in and can relate to makes them less reluctant to learn programming than if they just see it as a barrier. For example, one could take a couple of large datasets linked by a common variable and ask students to examine the files in a text editor and "understand" the data. Then the students can be given code to produce a captivating visualisation (e.g., Figure 14.3) with relevant insights for the subject matter of their course. Focusing on the reward, rather than the trigger, can lower anxiety (Xiao et al. 2022) and hopefully inspire students to create similar outputs themselves.

Having fun reduces anxiety in a wide variety of contexts, so why not try it for coding anxiety, too? Coding need not be boring! There are many resources that one can use to showcase how coding can be fun. Having fun coding will mean that when coding becomes required, students' receptivity to it is increased. R packages like `memer` and `mermery` allow meme creation from within R, while generative art packages can produce abstract paintings from code (Figure 14.4). The R package `fun` makes games and activities accessible from within the R environment. As an example, the classic game Minesweeper can be played within R via a single line of code defining the size of the area and the number of bombs. All the above can be used to introduce, in an informal setting, the concepts of functions and arguments. At the end of a long coding session, or even during one, the package praise can also come in handy. A call to its single function `praise()` returns positive feedback like "You are wonderful!" or "You are supreme!". These simple statements often put a smile on students' faces, and based on our experience, that is halfway towards reducing anxiety. Finally, if students just can't code but want to "fake it till they make it", the R function `look_busy()` in the `player` package will produce output on the console that will make anyone else believe students are working hard (showcase it, but discourage its use!). We have used several of these approaches and provide them as a starting point, but given R is a dynamic and ever-evolving environment, other fun and engaging options must live out there waiting to be used by teachers to help reduce code anxiety.

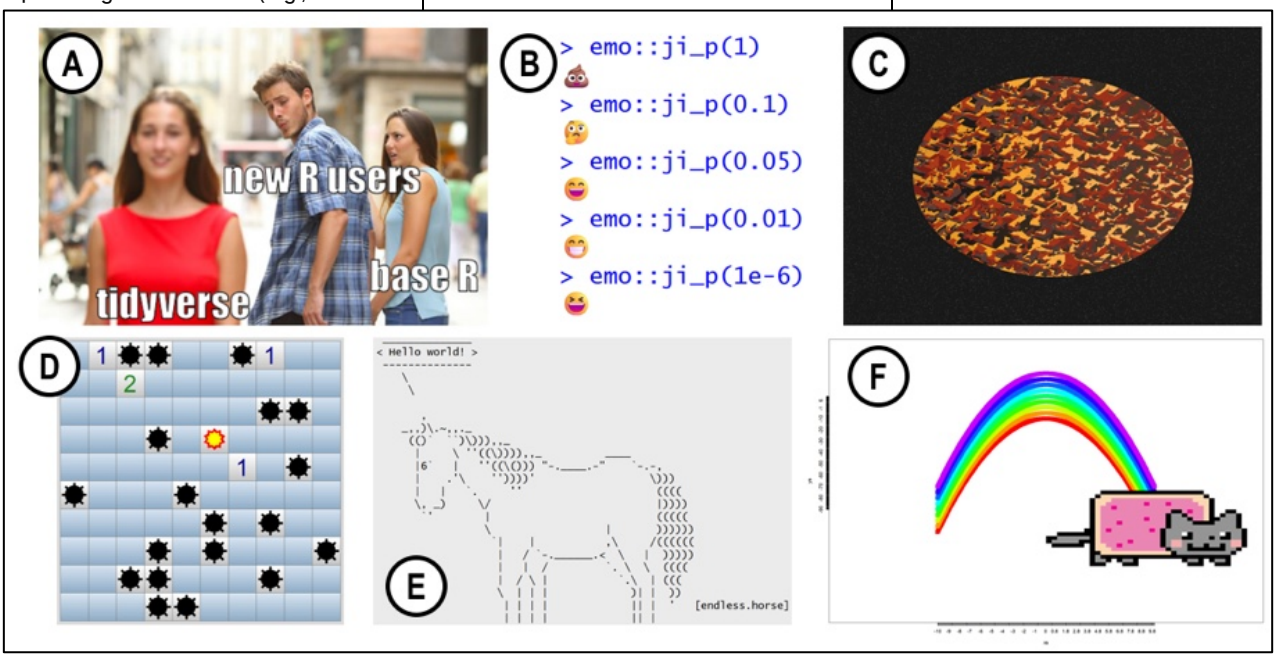


Figure 14.4: Examples of how R coding can be fun. A: A meme showcasing how memes can reduce code anxiety; this meme can be recreated with 2 lines of code using package memer[71]. B: illustrating P-values via emogi with package emo[72]. C: A painting created in aRtsy[73], a generative art R package. D: An instance of a minesweeper game created in package fun[74], demonstrating the authors inability to play it. E: An example of an animal that can be drawn in R syntax using the cowsay[75] package. F: The CatterPlots[76] package can be used to generate scatterplots that include cats.

[71] https://github.com/sctyner/memer

[72] https://github.com/hadley/emo

[73] https://cran.r-project.org/web/packages/aRtsy

[74] https://cran.r-project.org/web/packages/fun

[75] https://cran.r-project.org/web/packages/cowsay/index.html

[76] https://github.com/Gibbsdavidl/CatterPlots

In our opinion, live coding is one of the most effective tools to demystify programming and reduce coding anxiety. When instructors write, run, *and debug*, code in real time, narrating their thought process as they go (and getting error messages), they model both the practice and the mindset of coding. Students witness that even experienced coders make mistakes, encounter errors, and debug them systematically. This transparency transforms coding from a mysterious, high-stakes activity into a visible process of exploration, reasoning, and iteration.

Importantly, the approach taken during live coding matters as much as the content. Speaking aloud while typing, e.g., explaining why particular commands are chosen, predicting outputs before running them, and pausing to invite student predictions, turns what might otherwise be a passive demonstration into an interactive learning experience. When an error appears, embrace it as an opportunity to model problem-solving, showing that mistakes are normal, and even useful. Seeing how to respond to an error message with curiosity rather than panic can profoundly reframe students' emotional response to failure.

To lower anxiety further, live coding should begin with simple, visually rewarding tasks. Plot a small dataset, calculate a mean, or clean a short table. This way, students can follow the logic without feeling overwhelmed by syntax. Gradually, the exercises can build in complexity, moving from replication to modification and finally to creative application. When possible, students can be invited to suggest the next line of code, predict what will happen if a parameter is changed, or even take over the keyboard for short stretches.

Finally, recording or providing annotated versions of live coding sessions allows students to revisit material at their own pace. This ensures that the spontaneity of live teaching is complemented by the reassurance of reviewable, structured notes. By combining openness, interaction, and a healthy acceptance of mistakes, live coding transforms what might otherwise be a source of anxiety into a collective, confidence-building experience.

There can be multiple approaches to teaching code and statistical data analysis, and some of these might improve or contribute to coding anxiety (Young et al. 2026). Conventional wisdom suggests that the first step is the hardest for any journey. In our experience, starting with small, manageable, and clearly described tasks, will make the initial experience with coding positive, which will in turn decrease anxiety levels when facing harder tasks. This is consistent with the strategy suggested by (Auker and Barthelmess 2020). Later classes or assignments may contain less detailed guidance or be more open-ended, allowing students to explore their new found confidence in their coding abilities at their own pace.

Support and Collaboration Beyond the Classroom

Coding confidence grows most effectively outside of formal teaching time. Low-pressure practice, whether collaborative or individual, helps students reinforce skills and approach coding with curiosity rather than fear. This can be supported by structuring opportunities for independent study and peer interactions that sustain progress between classes. One simple but effective technique is to encourage students to maintain a short 'coding diary', allowing for normalisation of occasional frustration and celebration of key achievements.

The fear of revealing a lack of coding ability can be a powerful barrier, discouraging students from even attempting new exercises and progressing in their learning. Independent work outside the classroom, supported by carefully developed additional materials, can help overcome this. Providing a list of such resources will empower students to learn on their own. In our experience, a particularly interesting resource comes via the **R** package swirl[77]. Swirl "teaches you R programming and data science interactively, at your own pace, and right in the **R** console!". Contents can be explored independently, or by following the prompts of a 'virtual tutor' (Evkaya et al. 2026). Students are amused by a virtual tutor calling them by their name, meaning that they engage with the learning experience in a positive mood.

Sharing the learning journey with others can be equally powerful. The mantra "a problem shared is a problem halved" can be operationalised while teaching programming via pair-programming (Orzechowski, Blankinship, and Banas 2026). Pair-programming (Figure 14.5) is a useful strategy to lower coding anxiety levels (Fan et al. 2025). Nonetheless, these are not universal findings; for example, (Krizsan and Lambic 2024) found that while performance improved, it did not reduce anxiety levels. Online pair-programming might be less intimidating for some, although it is not clear whether the benefits associated with live pair-programming apply (Hafeez et al. 2023).

AI-assisted pair-programming, where a coder is paired with an AI agent, has become an accessible alternative to human-human pair-programming (Fan et al. 2025). This might be easier for students who would otherwise be reluctant to expose their difficulties to a colleague. While AI-assisted pair-programming has been shown to enhance motivation, reduce anxiety, and even improve performance, it does not fully match the collaborative depth and social presence achieved through human-human pairing (Fan et al. 2025). Regardless of the pair-programming mode considered, we believe that if well thought out and prepared, pair-programming activities can be used to lower coding anxiety levels in a range of students.

[77] https://swirlstats.com/

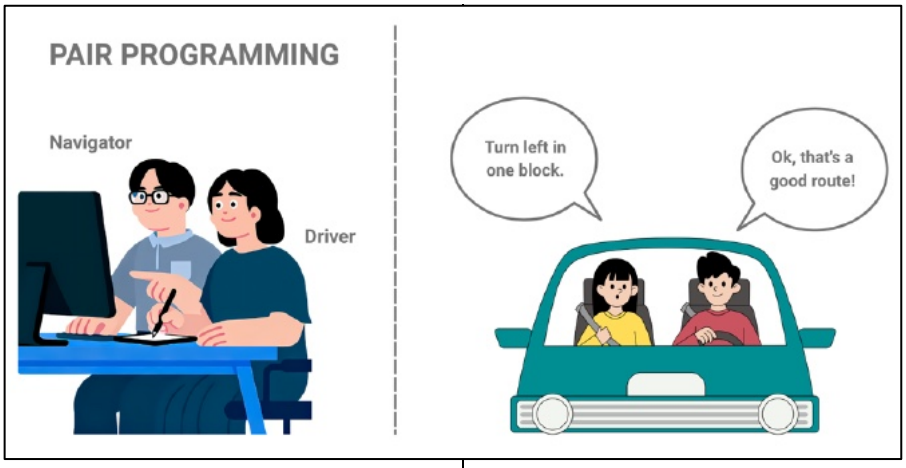


Figure 14.5: Pair programming and the analogy of a driver and a co-pilot

Students can be encouraged to further their learning journey by 'teaming up' with peers to continue to pair-programme outside of the classroom environment. This can help alleviate coding anxiety both through additional assistance in a familiar mode and also through the mechanism of shared endurance and a head-on approach to the anxiety barrier.

Ultimately, support beyond the classroom requires fostering autonomy and connection in equal measure. When students know they can access help, whether from peers, teachers, lecturers, or AI tools, they are less likely to interpret difficulties as personal failings. Instead, they begin to view coding as a shared and iterative process that need not come with coding anxiety but natural failure and success routines.

Key Takeaways

Coding anxiety is real, especially in students for whom coding isn't a core part of their disciplinary focus. This barrier should be recognised, and we have both a responsibility and an opportunity to normalise it in teaching settings and explain why and how the anxiety cycle can be broken. Overcoming coding anxiety should be presented as a personal endeavour, but one that instructors and peers, along with a growing number of resources, can support. For a teacher, effective methods for breaking the anxiety cycle are as pedagogically important as content creation. Instructors can become "anxiety diffusers", starting by exploring ways to provide rewards from coding activities. In particular, activities that allow students to see the potential and value of coding to achieve tasks they are inherently interested in should be considered. Providing encouragement and additional resources for learning how to code, and promoting additional high-rewarding coding activities, are likely to help further. Strategies to share the burden, like pair-programming, could further boost self-confidence in coding. Overall, different audiences and topics will require slightly different approaches, so engaging with students and intervening using the strategies discussed above will be fundamental for reducing coding anxiety and improving learning.

Funding

TAMs work is partially funded by national funds through FCT – Fundação para a Ciência e a Tecnologia, I.P., under CEAUL Research Unit, UID/00006/2025, DOI: https://doi.org/10.54499/UID/00006/2025

References

Auker, Linda A., and Erika L. Barthelmess. 2020. "Teaching r in the Undergraduate Ecology Classroom: Approaches, Lessons Learned, and Recommendations." *Ecosphere* 11 (4). https://doi.org/10.1002/ecs2.3060.

Bennett, Sue, Karl Maton, and Lisa Kervin. 2008. "The 'Digital Natives' Debate: A Critical Review of the Evidence." *British Journal of Educational Technology* 39 (5): 775–86. https://doi.org/10.1111/j.1467-8535.2007.00793.x.

Chua, Siew Lian, Der-Thanq Chen, and Angela FL Wong. 1999. "Computer Anxiety and Its Correlates: A Meta-Analysis." *Computers in Human Behavior* 15 (5): 609–23. https://doi.org/10.1016/S0747-5632(99)00039-4.

Connolly, Cornelia, Eamonn Murphy, and Sarah Moore. 2009. "Programming Anxiety Amongst Computing Students–a Key in the Retention Debate?" *IEEE Transactions on Education* 52 (1): 52–56. https://doi.org/10.1109/TE.2008.917193.

Craske, Michelle G., Michael Treanor, Christopher C. Conway, Tomislav Zbozinek, and Bram Vervliet. 2014. "Maximizing Exposure Therapy: An Inhibitory Learning Approach." *Behaviour Research and Therapy* 58: 10–23. https://doi.org/10.1016/j.brat.2014.04.006.

Evkaya, Ozan, Hebatallah Shoukry, TJ Elmas, Laila Dabab Nahas, and Steven Watterson. 2026. “An Optimistic Outlook on Teaching, Learning and Assessment for Coding with the Emergence of Generative AI.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Fan, Guangrui, Dandan Liu, Rui Zhang, and Lihu Pan. 2025. “The Impact of AI-Assisted Pair Programming on Student Motivation, Programming Anxiety, Collaborative Learning, and Programming Performance: A Comparative Study with Traditional Pair Programming and Individual Approaches.” *International Journal of STEM Education* 12 (1). https://doi.org/10.1186/s40594-025-00537-3.

Forrester, Chiara, Shane Schwikert, James Foster, and Lisa Corwin. 2022. “Undergraduate r Programming Anxiety in Ecology: Persistent Gender Gaps and Coping Strategies.” *CBE–Life Sciences Education* 21 (2): ar29. https://doi.org/10.1187/cbe.21-05-0133.

Hafeez, Mustafa, Anand Karki, Yara Radwan, Anis Saha, and Angela Zavaleta Bernuy. 2023. “Evaluating the Efficacy and Impacts of Remote Pair Programming for Introductory Computer Science Students.” In *Proceedings of the 25th Western Canadian Conference on Computing Education*. ACM. https://doi.org/10.1145/3593342.3593351.

Hofmann, Stefan G, and Aleena C Hay. 2018. “Rethinking Avoidance: Toward a Balanced Approach to Avoidance in Treating Anxiety Disorders.” *Journal of Anxiety Disorders* 55: 14–21. https://doi.org/10.1016/j.janxdis.2018.03.004.

How’s Life for Children in the Digital Age? 2025. OECD Publishing. https://doi.org/10.1787/0854b900-en.

Kampmann, Isabel L., Paul M. G. Emmelkamp, and Nexhmedin Morina. 2018. “Does Exposure Therapy Lead to Changes in Attention Bias and Approach-Avoidance Bias in Patients with Social Anxiety Disorder?” *Cognitive Therapy and Research* 42 (6): 856–66. https://doi.org/10.1007/s10608-018-9934-5.

Kaur, Tarandeep, and Samantha Newell. 2024. “The Silent Struggle: Experiences of Non-Native English-Speaking Psychology Students.” *Australian Journal of Psychology* 76 (1): 2360983. https://doi.org/10.1080/00049530.2024.2360983.

Krizsan, Tibor, and Dragan Lambic. 2024. “Examining the Impact of Pair Programming on Efficiency, Motivation, and Stress Among Students of Different Skills and Abilities in Lower Grades in Elementary Schools.” *Education and Information Technologies : Official Journal of the IFIP Technical Committee on Education* 29 (18): 25257–80. https://doi.org/10.1007/s10639-024-12859-w.

Meinhardt-Injac, Bozana, and Carina Skowronek. 2022. “Computer Self-Efficacy and Computer Anxiety in Social Work Students: Implications for Social Work Education.” *Nordic Social Work Research* 12 (3): 392–405. https://doi.org/10.1080/2156857X.2022.2041073.

Miller, Ainsley, and Kate Pyper. 2023. “Anxiety Around Learning r in First Year Undergraduate Students: Mathematics Versus Biomedical Sciences Students.” *Journal of Statistics and Data Science Education* 32 (1): 47–53. https://doi.org/10.1080/26939169.2023.2190010.

Mkrtchian, Anahit, Jessica Aylward, Peter Dayan, Jonathan P Roiser, and Oliver J Robinson. 2017. “Modeling Avoidance in Mood and Anxiety Disorders Using Reinforcement Learning.” *Biological Psychiatry* 82 (7): 532–39. https://doi.org/10.1016/j.biopsych.2017.01.017.

Nolan, Keith, and Susan Bergin. 2016. “The Role of Anxiety When Learning to Program: A Systematic Review of the Literature.” In *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*, 61–70. Koli Calling ’16. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/2999541.2999557.

Orzechowski, Pawel, Brittany Blankinship, and Kasia Banas. 2026. “Where Do i Even Start with Pair Programming in My Classroom? A Conversation with Seasoned Practitioners.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

PAN, YING, and Jamalludin Harun. 2025. “Conquering Coding Fears: A Systematic Review of Programming Anxiety in Higher Education.” *Journal of Information Technology Education: Research* 24: 020. https://doi.org/10.28945/5543.

Xiao, Pei, Liang Chen, Xiaoqin Dong, Zhiya Zhao, Jincong Yu, Dongming Wang, and Wenzhen Li. 2022. “Anxiety, Depression, and Satisfaction with Life Among College Students in China: Nine Months After Initiation of the Outbreak of COVID-19.” *Frontiers in Psychiatry* 12. https://doi.org/10.3389/fpsyt.2021.777190.

Young, Robert S., Rebecca L. Colquhoun, Tiago A. Marques, Brittany Blankinship, Ozan Evkaya, and William P. Kay. 2026. “Sequential Vs. Simultaneous: Approaches to Learning Programming and Statistics.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Zeidner, Moshe. 1991. “Statistics and Mathematics Anxiety in Social Science Students: Some Interesting Parallels.” *British Journal of Educational Psychology* 61 (3): 319–28. https://doi.org/10.1111/J.2044-8279.1991.TB00989.X.

3 Stars and 1 Wish: Small and Frequent Student Reflections Promote a Sense of Wonder and a Community of Vulnerability

Pawel Orzechowski

Edinburgh Medical School, University of Edinburgh
pawel.orzechowski@ed.ac.uk

Elaine Mowat

Edinburgh Medical School, University of Edinburgh

Karim Rivera Lares

Edinburgh Medical School, University of Edinburgh

Rebecca Sewell

Health New Zealand | Te Whatu Ora

Clare Llewellyn-MacRae

School of Social and Political Sciences, University of Edinburgh

Beatrice Alex

School of Mathematical and Computer Sciences, Heriot-Watt University

Kasia Banas

Edinburgh Medical School, University of Edinburgh

What Is This Chapter About?

The science of learning indicates that as students learn, they find it beneficial to reflect on what they understand, and to have a moment to check in with themselves (Van den Boom, Paas, and Van Merrienboer 2007). In this chapter we introduce **"3 Stars and 1 Wish" (3s1w)**, a technique for quick and frequent reflection. We value this approach and use it extensively with our students. We explain how it combines **self-reflection, knowledge scaffolding,** and how it helps to **build a community of learners.** You'll read reflections from students and teachers on how we've used 3s1w over the past six years, with over 1,000 students across more than 30 courses, representing a wide range of backgrounds.

In the **3 Stars and 1 Wish** process, students periodically write a very short reflection after every few hours of learning. Each student writes 4 short sentences: 3 things they consider interesting (stars) and 1 thing they wish to understand better (wish). To encourage transparency, where possible everyone's writing is shared. It contributes a small fraction of the final mark to stimulate participation and to reward effort. Below, we describe the motivations behind 3s1w, various iterations of the method, and how our colleagues are adopting and adjusting this technique.

3 Stars and 1 Wish in Education

Where Did It Come From?

The origins of 'Three stars and a wish' (or sometimes 'Two stars and a wish') are difficult to pinpoint, but the technique appears in magazines and reports for primary school teachers (Mullane 2013). It is used as a structure for children to talk about their achievements (stars) and goals (wishes) in relation to a particular subject, or to share their reflections on the week as part of a Friday classroom routine. It has also been used in schools and higher education as a constructive feedback technique in which positive points are followed by one area for improvement (Webb and Jones 2009; Wahid and Khan 2025).

What Does a 3 Stars and 1 Wish Post Look Like?

3s1w is a short four-sentence piece of writing which students produce right after learning something. The prompt that students get can look like this:

A Typical 3s1w Prompt For Students
Describe your personal response to this material in three short "star" sentences and one short "wish" sentence: 3 x Star: a thing that brought you joy or wonder, which you found new or interesting 1 x Wish: a thing you wish to understand better or to improve.

The format is very light-touch and students can write about anything, and in any way. Indeed, each cohort develops a slightly different culture of writing 3s1w. To give you an example, here's an example of a (fictional) 3s1w post from a Bird Physiology course. Many real students' submissions follow this length, granularity, and tone, but also there is a lot of diversity in writing styles. This is a 'check-in' of a student's own learning, so they write the way that works for them.

> 🌟 Birds' bones are are empty inside but very dense, which makes them sturdy but not really lighter!
>
> 🌟 Birds are very warm (40'C) and lower their temperature in winter (but do not hibernate)
>
> 🌟 Birds have great vision (including UV light) and possibly sense the magnetic field of the planet with their eyes, too.
>
> ❓ I would love to know how birds sleep. Do they keep flying? Do they have an autopilot to nap mid-flight?

Students submit their 3s1w reflections as discussion forum posts, one for each course topic (Figure 15.1).

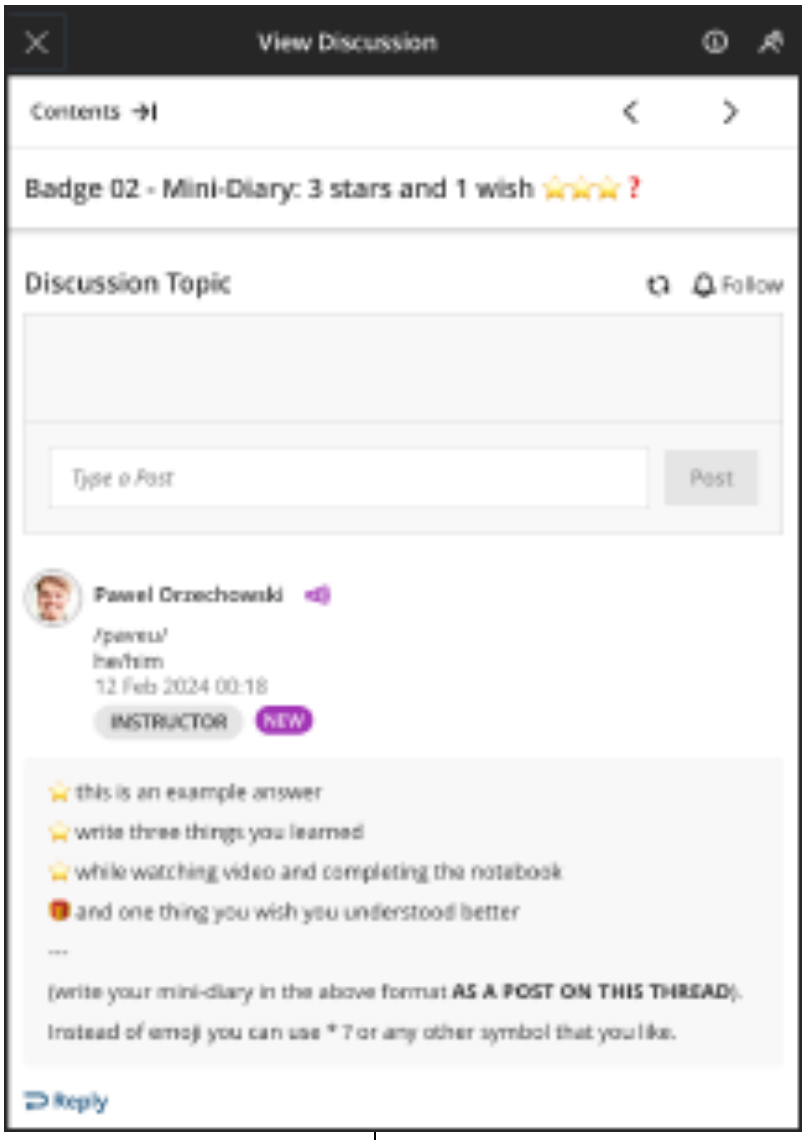


Figure 15.1: Instructors start a thread for each topic with an example first post. Students write their reflections below. This course was split into 15 badges, hence requiring 15 3s1w posts from each student.

Different Types of 'Wish'

Below are some examples of what students write about in the 'wish' section (these are fictional, but based on our observations). Notice that the points made are not only about the course content, but also about wider issues that the course made students think about. Students often write about their own practice and/or a change of perspective that the course content caused for them (Marshall et al. 2022).

> ❓ In response to the essay about communication styles, I find that I think more now about how to talk TO my superiors, not just hear FROM them. I am looking forward to seeing how this will develop in my practice in the future.
>
> ❓ In my hospital wing (where I work) we tried to accommodate visitors with dyslexia better. I wish everyone could watch this lecture to understand how important accessibility is, and that it does not have to be complicated or expensive.
>
> ❓ I wish I could access the slides for Wednesday's speaker. I loved their case studies, but was too immersed to take note of the details. Also more Q&A time with her would be great!

While we do not encourage using 3s1w to provide course feedback, or immediate requests for help, it occasionally happens. The wish to understand something better may stem directly from the fact that course delivery was less than ideal; for example, when the audio quality or text formatting made it difficult to understand slides, getting these things right could become a wish.

If you don't want the unique function of a wish to be hijacked by course feedback or requests for help, we recommend providing other means for students to achieve these aims. On most courses we have three separate discussion forums in addition to 3s1w: one for questions about the content; one for questions about course admin; and one for raising bugs. Every day a teaching assistant responds to student posts on those forums and engages with student questions there. This promotes understanding that requests for help or course issues should happen on these discussion forums, rather than in 3s1w.

Why We Adapted 3s1w for University Courses

Let us briefly introduce the reasons why we adapted 3s1w for our programming courses. Later in this chapter we will expand on the topics of motivation, community and authentic (inward-focused and world-focused) learning.

As educators, we value the use of assessment as a tool for learning, rather than just a mechanism for giving students marks.

We created 3s1w so that it gives an opportunity/excuse to learn more, as an "assessment for learning" (not

"assessment of learning") (Black et al. 2003; Wiliam 2011).

We also believe in allowing the community of learners to lead the course experiences, rather than focusing on lone work, or on rigid structures imposed by the teachers. We drew inspiration from the concept of 'Liberating Structures' (Lipmanowicz and McCandless 2013), a collection of lightly-constraint activities designed to allow learners to be in charge of their learning. Each cohort of students shape their 3s1w forums slightly differently, and we do not interfere as long as they share their insights equitably and kindly.

We also take inspiration from the concept of a Gratitude Journal (Hebbink, Ruyter, and Schinkel 2025) where participants are regularly asked to list things they are grateful for. This is meant to have an effect of motivating them to 'tune into' noticing good things, so that they have something to write about. The same can happen as learners are looking out for things that bring them joy in the course. Since students see each other's posts, we also observed a 'Community of Vulnerability' – a space where students feel safe to admit to their struggles, and see that they are not alone. The "wish" aims to replace the fear and stigma of being confused or lost, with a sense of pride in the educational struggle (Kapur 2008). Moments of struggle, and developing strategies to overcome it, are an important part of learning. This approach promotes sustainable growth over maintaining a false facade of constant high achievement.

We have been using 3s1w as a short, high frequency, reflective learning activity within university courses at the University of Edinburgh. Between 2019 and 2026, over 1000 students have participated in 3s1w activities as part of 30+ courses delivered across the Business School, the Edinburgh Futures Institute, and the Edinburgh Medical School. The courses included online and onsite delivery, postgraduate and undergraduate curriculum, and have ranged from 20 to 150 students.

Let's Reflect on (Apparent) Problems With Self-Reflection:

There's plenty of evidence that reflection helps learning (Boyd and Fales 1983): it consolidates knowledge, strengthens critical thinking, and gives learners self-awareness of the learning process. On the flip side, if done badly, reflection can be confusing, time-consuming, intimidating, lonely, and unrewarding. It can become a cringeful chore if we're forced to do it and we don't see the point. Doing it because we have to and not because we see it as useful can lead to a dishonest reflection, or worse, outsourcing it to LLMs (we term this: 'cheating yourself out of learning'). We have a responsibility as teachers to encourage students to reflect, but it has to be done honestly, thoughtfully, and most importantly, usefully.

It's Confusing

Have you noticed that some students, and even teachers, do not have a clear idea of what "self-reflection" means? Are you aware that it may not have an unblemished reputation? Why is it that reflecting, or thinking about our actions, experiences and thoughts, might have developed a bad reputation? For some, reflection sits on the same shelf as meditation, therapy, or positive thinking. These are all useful actions that help us understand ourselves and how we interact with the world around us, but they can feel jarring when we're forced to do them. This is especially true if we don't understand why it is useful, if it takes a long time, and if we don't really understand what we are supposed to do. Often, our first experience of doing this is at school; therefore, the request to reflect may trigger long-held resistance.

Following a suggestion from our students, we reframed reflection as a 'check-in'. This is a moment in time when you take a breath, take stock of the things you have learned, how they sit with you, and how they make you feel. Learning is accompanied by a cycle of strong emotions: joy and wonder; occasionally struggle, frustration and determination; and finally victory and hunger for more. The types and intensity of these feelings will depend very much on the person and what they are learning. Students will feel different things at different points in the process, and these feelings will not perfectly align across the cohort. We suggest, therefore, that the aim of reflection within a course should be to create multiple, flexible moments in the learning process where all these feelings will have their place, and to recognise that both positive and negative feelings are a normal part of the learning process.

Thoughtful consideration is needed if teachers want to include reflection as a part of the course assessment. Ideally, we would like reflection to be part of an authentic assessment practice. "Authentic assessment" can be interpreted in different ways: sometimes it means "work-like", or "using real data/challanges", or "anchored in the context of the course". For us, authentic means **honest and deeply anchored in the students' positionality.** We want to know: who they are, how they think, and what mental models and experiences they bring to the cohort. In 3s1w, students do not make a shopping list of what they learned; instead, they outline what mattered to them as a human. The truly authentic assessment is one where each student can share their own unique and honest perspective of their learning experience.

It's Time Consuming

Often, when we ask students to reflect on what they have learned, it is at the end of the process. We want them to reflect on everything they have been told, all the interactions they have had, and the multiple perspectives they have seen in that process. This type of extensive reflection takes time. In introducing 3s1w, we craved a learning activity that we could do little and often, something that takes just a few minutes to do, but gives students an excuse to recall what they have learned. Not just to make a list of main topics or headings, but to be able to really anchor it in their own experience,

previous knowledge, and learning journey. We wanted them to reflect often, so they could focus on a specific recent piece of learning. We wanted their gut reaction and to know what they really thought in the moment.

A significant feature of 3s1w is that if it is done well, it comes almost for free - it barely requires extra time. Students are likely to be taking notes as they watch videos, read articles and complete exercises - the 3s1w gives them the opportunity to capitalise on these notes whilst still retaining the context and feelings of that process. They get "paid again" for the same work. It encourages being mindful and present while learning, looking out for something that they find interesting, because they know that in a minute they will have to list three interesting things and one thing they wish they understood better.

This process, if done often, is very quick. We encourage students to revisit their notes frequently, for example, a check-in every 2 hours during a learning activity, and to extract and explain the parts that impacted them the most. We have read tens of thousands of reflections in 3s1w entries, and we see that what gets recorded is not necessarily the most important or difficult aspects, but rather it is the things that students found most surprising or interesting. What we found caused us to evaluate what and how we teach in a new light, looking at our course materials in a new way, and from a fresh angle. This new perspective allows us to refocus, become better teachers, and give students new benefits from the work they've already done.

It's Intimidating

For teachers, asking our students what they find interesting may be intimidating, we might think "but what if they did not find anything interesting in the class?". This either means that we did not do a very good job, or that the students did not pay attention. In either case, we want to know about this, and we want to know early, so that we can adapt our teaching style. In our courses, when we use 3s1w, we pay attention to the granularity and frequency of check-ins. As we encourage immediate reflection, students are asked to reflect more rather than less often, for example, after every few hours of learning, after an activity, or as a weekly check-in on a long course. In other words, we ask students to reflect as soon as there is something to reflect upon, and to do it immediately afterwards, when their thoughts and feelings are still fresh.

For students in higher education, struggle is an important part of learning: we do not understand things until we have grappled with them. Reflection, necessarily, will contain what is still unclear and what the students want to understand better. Struggle can stimulate negative feelings, and they need somewhere to go. That's where the 'wish' part of this method comes in. There is a designated place to be vulnerable and to give oneself space to not be confident - yet. When a group of learners see that others are in the same position, that they feel that way too, it can bring reassurance and a sense of community where moments of struggle are shared. 3s1w gives students an opportunity to celebrate and be rewarded for their struggles as much as for their successes.

It's Lonely

The 'self' in 'self-reflection' clearly indicates it's something we do by ourselves - it involves introspection, tuning inwardly to what we feel and think. This is important, because at the end of the day we can't outsource our own learning and growth, no-one can do it for us, (despite what the marketing of AI corporations tells you). Once we have done our thinking and reflecting and externalised it as a piece of writing, we are standing at the well of social learning with a bucket in our hand. Imagine the experience of reading what the other 100 students in your class found interesting - recognising your own stars but also noticing fascinating things that might have escaped you. Further, imagine the powerful feeling of sharing your doubts and wishes, you are no longer alone in the sometimes difficult pursuit of understanding, but you are part of the 'community of vulnerability'.

In our courses, we often implement 3s1w as an online discussion forum, where each student posts their stars and wishes after completing a section of the course. Students can read each other's posts, and each piece of writing contributes to a resource co-created by the cohort - a list of all the things that someone could find interesting or could struggle with. We even allow students to read other people's posts before they write their own. At face value, it is not obvious if it's a pedagogically good strategy, but we want to encourage students who might experience writer's block. As far as reflection goes, reading a number of other people's reflections and picking (and re-posting) those that personally spoke to you, can almost be as good as writing them yourself.

It's Unrewarding

Motivation is an intricate mix of internal and external reasons to do something. Intrinsic motivation can come from curiosity about the subject, engagement with the way teaching is delivered, or from gamification mechanisms. Gamification can motivate students by encouraging their desire(s) to: complete all tasks, to be the first one who at the finish line, or to see themselves as part of a collective team effort. Course-wide 3s1w discussion forums can utilise these game-like motivators positively and safely when combined with a meaningful code of conduct.

As teachers, we can also provide students with extrinsic sources of motivation, such as grades or the appreciation of their cohort community. Over the years, we have experimented with different ways in which 3s1w can contribute towards the course grade. In some courses it was completely voluntary, but in most courses, we have found it helpful to assign a contribution of 10% of the final grade. This is a reward for taking part in the process and encourages engagement. For example, in a course with 15 check-ins, each is worth 0.7% of the final grade, with a total of 10%. This is enough to motivate almost all students to submit their reflections, but also it is a small enough fraction as to not disrupt the final course marks. There is no 'correct answer' to a written reflection of this

type, it is by design very personal and individual; we give each 3s1w entry full marks if it appears to be written by a human and follows the 3s1w format. In the tens of thousands of student entries over the years, we have struggled to find any submissions that do not meet this criteria, no-one ever wrote "blah blah blah" as their entry! This is possibly due to students keeping each other accountable, as all posts are visible to the whole cohort. One student (see below) told us that she was not even tempted to cheat and use LLMs to write 3s1w for her, because writing the prompt would take her longer than just writing the 3s1w by herself. We consider this our biggest success: honest learning being faster and easier to do than cheating. This is the true meeting of 3s1w being 'almost for free'.

Main Themes From Feedback and Our Experience

Assessment FOR Learning (Not Just OF Learning)

Assessment is, for better or worse, a core part of university courses, serving purposes such as quality control, motivation, and student-differentiation (S. Brown and Knight 2012). But when grades and assessment become solely key performance indicators ("things to focus on"), students can over-optimise for getting good grades at the expense of deep and lasting learning. To save the baby from being spilled with the bathwater, course designers can harness the pressure and reward of assessment as a motivating factor.

To recycle (or compost) the idea of assessment we can relax on the **Assessment OF Learning** (mundane busy-work, to catch students out and take away points for what's missing) and strive towards an **Assessment FOR learning** (work which provides more opportunities to learn, while fitting in with the existing assessment frameworks) (Wiliam 2011). In 3s1w, a small reward (i.e., marks for any meaningful participation) for a simple activity becomes an excuse to engage in deeper learning.

Given the progress of cheating technology (LLMs) and pressures of life outside the university, the temptation to play the system grows, so a meaningful way to learn can be a motivational carrot, in a stickless world.

> "In terms of cheating, I think this particular assessment is not prone to using AI because it's so short. If you asked about more text, even a paragraph, I would see an appeal of using LLM to write or correct my writing. And instead, I'd read through other people's posts and use that as an inspiration. And that's a good thing, reading other people's posts felt like learning." - Rebecca, Postgraduate Student

Build Community, So That It Builds You

We meet friends for life when we are students, and both identities (friends and students) have the capacity to enhance the other one. Feeling part of a community is a meaningful but also challenging aspect of the higher education learning experience (Haddow and Brodie 2024). Building a sense of community may be especially important in online and blended learning, where 'social presence' - the degree to which online participants feel connected to one another - has been identified as a key factor for successful learning (Whiteside, Dikkers, and Swan 2023). When online or hybrid, learners may be missing out on all the small 'water cooler' interactions with fellow students, e.g. chats while waiting for the lecture, or discussing things in the cafeteria.

3s1w naturally builds community throughout a course by enabling students to connect with each other on a regular basis, encountering others' points of view, each other's successes and doubts. When going beyond enforced positivity, students start sharing 'wishes', it can foster a 'community of vulnerability' in which struggle becomes a source of connection, pride, and reward (B. Brown 2015).

> "It was good to see what people said they didn't understand. There was a feeling of community like we're all in the same boat." - Rebecca, Postgraduate Student

There are other benefits of reflection within a community. Seeing fellow students' reflections can be a way of spotting things one missed, being reminded of something that felt important at the time, or seeing something with fresh eyes.

> "People would often write about related topics they found interesting, or fun things that they'd like to check out afterwards. And often I'd be like 'oh I didn't think about that! It's interesting, I could go and read up on it myself too'." - Rebecca, Postgraduate Student

Pulse-Checking for Oneself and for the Cohort

Experienced educators can read the room (of 10 or 300 students) to estimate the level of energy and attention at their disposal. We skillfully build in breaks, or know when we can push the group a bit deeper into a challenge. But much of students' learning happens outside of contact hours - what can we do to not set them up for failure when later they continue their studying alone? One solution we adopted is to do the most challenging bits when we are together (practice), and then students do all the bits which need pauses and time to sink in by themselves (theory).

'Flipped classroom' is a course delivery method in which students first engage with learning materials individually, and then come to class ready for discussions and practical exercises (Al-Samarraie, Shamsuddin, and Alzahrani 2020). It 'flips' the narrative from 'listen together, struggle alone' to 'listen alone, practice together'. It works really well, also for diverse student cohorts, but it needs students to actually do the preparation tasks. A key ingredient for this to work is diligence - students need to come prepared, having absorbed the learning material and ready to work with others (Baig and Yadegaridehkordi 2023). If the accountability is not there, the flipped classroom will fail spectacularly.

When 3s1w is added to the flipped classroom, it creates a visible check-in point for students to reach after their individual preparation, and before we meet together to practice. This small and tangible token of 'I am done' gives each student the opportunity to self-regulate their learning. Since the posts are public within the cohort, they also create peer encouragement to write check-in notes that are meaningful for oneself, and for others.

> "It didn't feel like it was an assessment. It was more of a "check in". But since it was visible to other students, their opinion was my motivation to write meaningful things, not just gibberish." - Rebecca, Postgraduate Student

Students can see how they are doing and how their peers are getting on. Each student has a very clear record of how their own learning is progressing, knowing which parts they already completed, and which ones are still outstanding.

> "3s1w was an important part of my learning. I'd do each badge and then I'd use 3s1w as a checkpoint: to know what I've done already, and if I understood it. In Learn, it would get nicely ticked off, so I knew what I've done and what's next" - Rebecca, Postgraduate Student

Staff can also see how the cohort responds to the material and how the course is going. This is the closest thing to 'reading the room' we'll ever get for asynchronous learning.

The Piggybacked Feedback Rodeo

Although 3s1w is primarily about students' own learning and what they 'wish' they understood better, it is equally open to students using it to express a wish in relation to the course. In this way, 3s1w becomes a mechanism for rapid course feedback, functioning as a ready-made suggestions box. But there is a real danger of conflict of interest and mixed incentives here - 3s1w should be a tool for students to learn better, not for teachers (or managers) to have another surveillance mechanism.

We believe that piggybacking course feedback on 3s1w should not be encouraged. Any feedback or quality control mechanisms tend to self-erode following Goodhart's Law ('When a measure becomes a target, it ceases to be a good measure'). Unfortunately, higher education, as any industry, is susceptible to rigging and playing the system (Strathern 1997). When unchecked, the piggyback that was initially intended to give you a lift turns into a dangerous and counterproductive pig rodeo. By the way, the term piggybacking actually has nothing to do with riding pigs, but this was too good a metaphor to miss.

The Freeing Power of Minimal Constraints

With its simple four-sentence format, 3s1w represents a 'just enough' sweet spot of structure that is neither too inhibiting nor too loose. Indeed, once students know how the 3s1w technique works, they do not need teachers or supervision to keep using it - it is a structure designed to set them free, not to control. When we codified 3s1w for use on university courses, we were inspired by collection of 'Liberating Structures' (Lipmanowicz and McCandless 2013) - techniques to discuss and contribute equitably, with no-one in charge. Structures like those put very light (or none) demands on facilitation; once set up, little input is required to keep it running. This freedom means that it can scale into many students, and many check-in points per course. But also, like with every self-organising structure, everyone needs to buy into the code of conduct and the principles of kindness and curiosity.

The small, finite number of sentences in 3s1w is also a strength, forcing students to pick just three favourite things, and one space for improvement - it cannot be seven stars and three wishes. Just as a tweet, an elevator pitch, or a haiku poem forces the writer to clarify what they mean, having a tight writing limit can be beneficial, requiring students to pick out what is most meaningful. Writing less also reduces fatigue, and the bullet-point format allows simple (or indeed chaotic) grammar.

Writing what resonated with them (rather than what was important) also means that students can mark the learning as theirs, extracting what is special for them, shining a light on what they are most curious about. As observed by the filmmaker Werner Herzogin an interview[78]: "We are not garbage collectors [...] we are thieves. We get away with loot from the most spectacular [...] places you can ever find!"

> "Many posts had some genuine insights in them - I really liked those! Students were deeply reflecting, rather than just repeating and rephrasing. I especially liked when someone put took two pieces of material and related them to each other, or presented them from their own perspective. People wrote quite authentic reflections: how they felt about the topic; why it was interesting to them; did it make them think about." - Rebecca, Postgraduate Student

Tune In, Notice, and Let Your Words Create Worlds

We usually get better at the things that we do often. When students complete frequent 3s1w check-ins, they can build a habit of naturally identifying their stars and wishes as they go through the materials. If you play a game of spotting red objects, you will keep noticing red objects long after the game is over.

Just like in other regular awareness practices such as mindfulness or journalling, we get better at tuning into what we need. By 'deliberately turning attention inwards to explore the workings of one's own mind ... one's inner world of thoughts, sensations and emotions becomes part of the curriculum' (Ergas and Hadar 2023). It's not that we get better at writing four

78 https://www.youtube.com/watch?v=dE8PULVkeWM&t=75s

bullet-points, but we get better at noticing the things we could write about. As students tune into themselves, and into the learning moments as they happen, looking out for what is curious, interesting or frustrating, they may be embarking on an important 'inner journey' (Ergas and Hadar 2023) and a way of organising and anchoring learning that can filter into their lives more generally.

Indeed, this practice is more than just about training ourselves to see some version of a reality, but rather with our words and actions we get to shape a reality with, and for, our fellow students. Inspired by social constructionism, we see reality as subjective rather than objective; we create it together through language and conversation (Gergen 2015). We see 3s1w not just as a practice of writing bullet-points, but as an invitation to celebrate progress ('stars') and to share a hope or a desire ('wish'). So, students get to choose if they'd rather treat 3s1w as a chore and live in a world where learning is a drag; or live in a curious world full of wonder and quests where even a problem is an achievement.

Using 3s1w in Real Courses

Each course and each cohort are different. Below is a case study of one of our courses, followed by examples of what can be modified and what results we've seen.

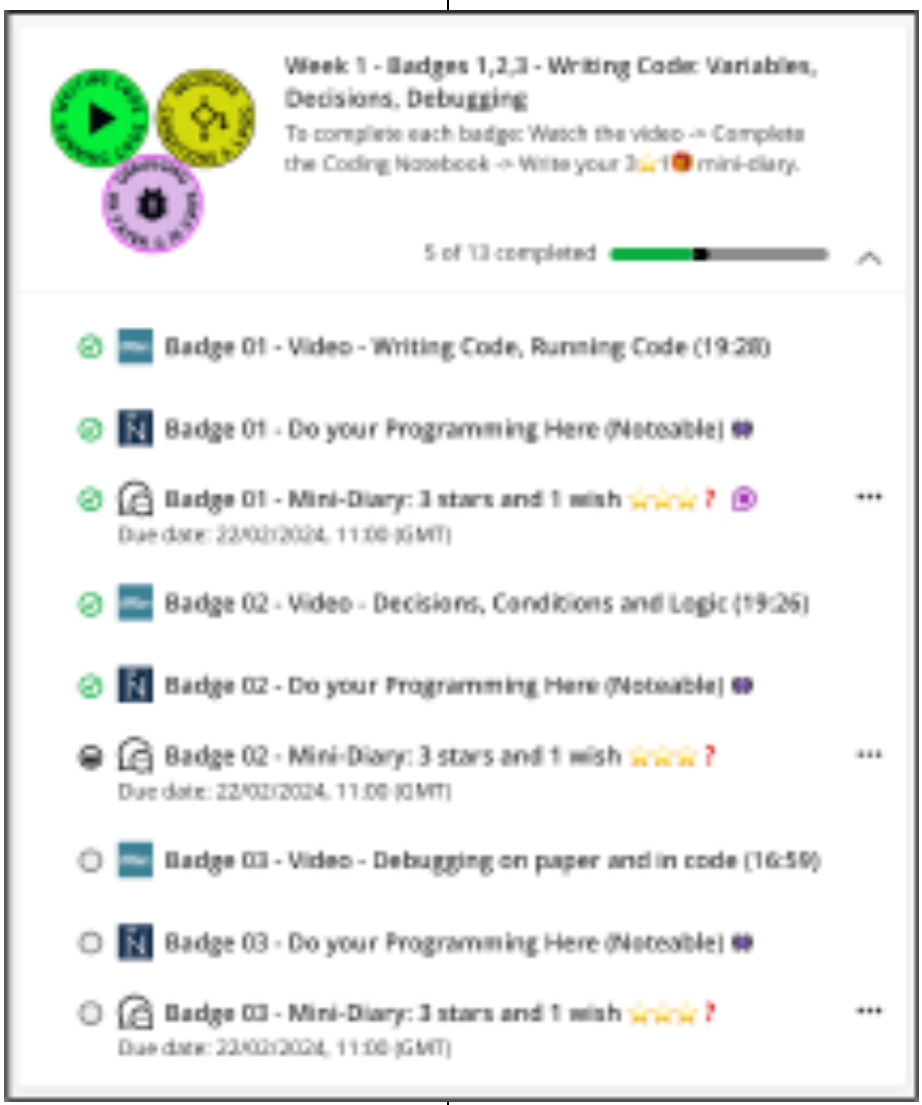
Case Study: Foundations of Software Development

3s1w has been used in our Foundations of Software Development course (in the Business School, and later in the Edinburgh Medical School). This course has been adapted and delivered to undergraduate and postgraduate students, in-person and online. Below, we will describe its use to illustrate the practicalities of using 3s1w in a university course.

In Foundations of Software Development, students learn in 1.5-hour long chunks called badges - each badge consists of a video, a coding notebook and a small exercise. Right after completing each badge, we ask the students to write their 3s1w on an online forum where it's visible to all other students. In each learning week, students are given 2-3 badges to complete (each ending with a 3s1w entry), which then adds up to about 15 entries over the entire course (Figure 15.2).

Participation in 3s1w is worth 10% of the final grade, so if a student meaningfully writes 9 out of 15 posts (60%), they would get 6 out of 10 points towards the course grade. Since this is a low-stakes assessment, each post that is on topic and within the 3s1w format gets full points. Participation is very high, with 80% of students completing all entries, and around 60% of students completing the entries as they learn (rather than 'cramming' them at the end). A member of the teaching team reads the entries and produces a 'weekly digest' email for students about what everyone enjoyed and struggled with that week.

Figure 15.2: Typical course week includes 3 Badges, each consisting of a video, exercises and 3s1w reflection. Student progress is denoted with ticks for gamification purposes. Here you see the week 1 of the Foundations of Software Development



Practical and Tested Considerations for Customising 3s1w

Teacher Workload: If points are given for participation (rather than content) then the marking load is lower, but most students achieve full marks. This may inflate the average grade. In terms of feedback, in most courses we do not respond to each individual post, but rather we read and respond to all posts as a 'weekly digest' email.

Contribution Towards the Final Grade: Assigning 30% of the final mark to 3s1w increased the final marks dramatically (not balanced); 0% still worked but decreased participation (roughly from 80% to 55% or students completing all posts); 10% seems to achieve the best balance between motivation and grades.

Authenticity and Cheating: Because students see each other's posts, no-one spams the forum by writing gibberish. Since each submission is short and there is no model correct answer, we see participation points as a valid strategy. This means if you write something, you get points. Cheating-wise, reportedly it is easier/faster to just write the 3s1w posts than to cheat using GenAI.

Openness: We experimented with if and when students can see the posts of everyone else in the class. There are three options: all students see all posts immediately; each student sees only their own posts; or each student 'unlocks' others' posts only after they post their own. There was a concern that if students can see others' writing before they write their own, they will plagiarise. But we believe the opposite is also true - seeing other people's posts will inspire and uncover the nuance of one's own reflections. We usually opt for the "everyone sees everything" strategy or occasionally "post yours to see other's posts".

Means of Submission: For all of our courses we use online forums inside of our Virtual Learning Environment (VLE), with the teacher setting up one forum thread per one topic. For in-person workshops we used post-it notes on the lecture room's door with one post-it fitting in one 3s1w entry.

Other Prompt Formats: The number of stars and wishes can be modified, as long as there are more positive than negative points required. Some colleagues have extended the method to include an action that the students can take to address the wish ("3 Stars, 1 Wish, and a Step") and to encourage expressing curiosity ("I Like, I Wish, I Wonder").

Frequency: We want to encourage students to capture authentic and emotional reactions to the learning content. This requires 3s1w to be written immediately after the learning episode, and so we have placed them after every 1-2 hours of learning. However, some colleagues have tried doing a weekly, or even one-per-course 3s1w^ *[^footnote: Something to beware is to not use this method as just another way to gather course feedback - it's for students to reflect and check-in with their own learning. See 'Piggybacked feedback rodeo' above.]*

Integrating With the Course and Reminders: On many courses we achieve very high completion rates, we suspect, due to the number of ways we remind students to write their 3s1w. Each programming notebook (completed immediately before it's time for 3s1w) ends with a prompt: "Now is the time to write your 3s1w". Weekly emails and the layout of the VLE are also structured to promote and remind students about writing their entries.

It Is Not a Silver Bullet: Occasionally we see courses where 3s1w does not work too well. There might be a number of reasons for it. We usually comb student feedback for clues, improvements, or reasons why a particular cohort of students on a particular course did not click with this method. We have not seen consistent deal breakers in terms of course topic, size or mode of learning, but the method has to be applied with care, curiosity and conviction. If it is done badly, it carries a risk of busy-work fatigue and other opportunity costs. Students will benefit more from going for a walk or having a nap, than from partaking in badly implemented 3s1w.

What Not To Forget When Incorporating 3s1w in Your Course Experience

Make It Part of the Course Structure

Some course activities, like note-taking, are great for learning, but might be considered auxiliary and optional by students. When students watch, read or complete exercises, the process of taking notes will help to clarify and anchor their learning (Salame, Tuba, and Nujhat 2024). With 3s1w students can 'get paid again' for their effort. Students can repurpose their notes by picking their favourite and most striking points for their 3s1w post. Repurposing is a smaller task than creating something from scratch, and in some sense it comes for free: most of the job has already been done.

If adopting this strategy, make sure to both feature 3s1w prominently and weave it into other course activities. It has the capacity to be the engine, the glue of the course, but it needs to be carefully and thoughtfully designed into the course as a whole. We have found that it works well when integrated seamlessly into the student workflow, including the course calendar, content, and platform. For example we feature 3s1w right under each section in our VLE, so that students know where it is.

> "Each badge consisted of: a video; an exercise sheet; and a 3s1w"check-in". 3s1w was the activity connecting all content together." - Rebecca, Postgraduate Student

Make 3s1w Congruent With Your Overall Course Culture

Congruency (or genuineness) has been identified as a core condition for effective communication and relationship-building (Rogers 1961). As such, it is arguably an important element in learning relationships too. For 3s1w to flourish it needs to be part of a programme or course context which conveys a sense of curiosity and care - someone is listening!

> "3s1w worked because of the great community feel fostered by the wonderful staff and students on my course. If 3s1w was just 'air

> dropped' into a course with lower engagement or trust, would this become a chore, instead of a place for vulnerability or thoughtfulness?" - Rebecca, Postgraduate Student

Make It Happen Immediately After Learning

Including 3s1w as a frequent activity that captures meaningful moments as they occur is key to its currency and interest for students. We recommend embedding it at regular points that make most sense for your particular course design, taking care to share the rationale with the students so they are clear about why they are doing it.

> "There is something great about asking immediately after completing some course content. The immediacy of what you like and what you didn't like... it's all very fresh in your head. You still remember things which were interesting and which you did not understand. It works great when it captures the thoughts and feelings as they happen. But I don't think 3s1w would work after some time, like a week later." - Rebecca, Postgraduate Student

The format is deliberately short to ensure the activity is easily do-able and to discourage overthinking or writing paralysis (Nurkamto et al. 2024).

Reward Participation Rather Than Performance

Rewarding participation rather than performance keeps 3s1w motivating and authentic for students, and much less complicated for markers.

> "Crucial part was that we got full marks for each attempt, rather than percent of mark based on performance. If it was being formative - less people would do it; if it was scrupulously graded - it would undermine the honesty of it." - Rebecca, Postgraduate Student

Now It's Your Turn

We hope that this chapter has helped you to consider **why** and **how** you might implement 3s1w within your own context. In terms of our own stars and wishes, we have learnt a great deal about implementing 3s1w, as outlined above. Our wish now is to understand even more about how to support reflection, celebrate learning and build community through this kind of shared activity. Let us know how you get on if you go on and implement 3s1w or to devise your own variation!

Why not finish the reading of this chapter by writing your own 3 stars and 1 wish about what you just learned? Here's some space to do it, so grab a pencil and give it a go!



References

Al-Samarraie, Hosam, Aizat Shamsuddin, and Ahmed Ibrahim Alzahrani. 2020. "A Flipped Classroom Model in Higher Education." *Educational Technology Research and Development* 68 (3): 1017–51.

Baig, Maria Ijaz, and Elaheh Yadegaridehkordi. 2023. "Flipped Classroom in Higher Education: A Systematic Literature Review and Research Challenges." *International Journal of Educational Technology in Higher Education* 20 (1): 61. https://doi.org/10.1186/s41239-023-00430-5.

Black, P., C. Harrison, Clare Lee, B. Marshall, and D. Wiliam. 2003. *Assessment for Learning: Putting It into Practice*. McGraw-Hill Education (UK).

Boyd, Evelyn M, and Ann W Fales. 1983. "Reflective Learning: Key to Learning from Experience." *Journal of Humanistic Psychology* 23 (2): 99–117.

Brown, Brené. 2015. *Daring Greatly: How the Courage to Be Vulnerable Transforms the Way We Live, Love, Parent, and Lead*. Penguin.

Brown, Sally, and Peter Knight. 2012. *Assessing Learners in Higher Education*. Routledge.

Ergas, Oren, and Linor L Hadar. 2023. "Does Mindfulness Belong in Higher Education?–an Eight Year Research of Students' Experiences." *Pedagogy, Culture & Society* 31 (3): 359–77. https://doi.org/10.1080/14681366.2021.1906307.

Gergen, Kenneth J. 2015. "From Mirroring to World-Making: Research as Future Forming." *Journal for the Theory of Social Behaviour* 45 (3): 287–310. https://doi.org/10.1111/JTSB.12075.

Haddow, Christine, and Jacqueline Brodie. 2024. "Harnessing Innovation Approaches to Support Community and Belonging in Higher Education." *Innovations in Education and Teaching International* 61 (6): 1341–54.

https://doi.org/10.1080/14703297.2023.2176907.

Hebbink, Nick, Doret de Ruyter, and Anders Schinkel. 2025. “The Educational Value of Gratitude.” *Educational Theory* 75 (5): 802–22. https://doi.org/10.1111/edth.70035.

Kapur, Manu. 2008. “Productive Failure.” *Cognition and Instruction* 26 (3): 379–424.

Lipmanowicz, Henri, and Keith McCandless. 2013. *The Surprising Power of Liberating Structures: Simple Rules to Unleash a Culture of Innovation*. Liberating Structures Press Seattle, WA.

Marshall, Tony, Saskia Keville, Alison Cain, and Joanna R Adler. 2022. “Facilitating Reflection: A Review and Synthesis of the Factors Enabling Effective Facilitation of Reflective Practice.” *Reflective Practice* 23 (4): 483–96. https://doi.org/10.1080/14623943.2022.2064444.

Mullane, Jayne. 2013. “Three Stars and a Wish.” *Primary Teacher Update* 2013 (27): 34–35. https://doi.org/10.12968/PRTU.2013.1.27.34.

Nurkamto, Joko, Lastika Ary Prihandoko, NHPS Putro, and Oikurema Purwati. 2024. “Academic Writing Apprehension in Higher Education: A Systematic Review.” *Studies in English Language and Education* 11 (1): 247–66. https://doi.org/10.24815/siele.v11i1.28928.

Rogers, Carl R. 1961. “On Becoming a Person. Boston: Houghton Mifflin.”

Salame, Issa I, Maryam Tuba, and Mir Nujhat. 2024. “Note-Taking and Its Impact on Learning, Academic Performance, and Memory.” *International Journal of Instruction* 17 (3): 599–616. https://doi.org/10.29333/iji.2024.17333a.

Strathern, Marilyn. 1997. “‘Improving Ratings’: Audit in the British University System.” *Europaeum Review* 5 (3): 305–21. https://doi.org/10.1002/(SICI)1234-981X(199707)5:3<305::AID-EURO184>3.0.CO;2-4.

Van den Boom, Gerard, Fred Paas, and Jeroen JG Van Merrienboer. 2007. “Effects of Elicited Reflections Combined with Tutor or Peer Feedback on Self-Regulated Learning and Learning Outcomes.” *Learning and Instruction* 17 (5): 532–48. https://doi.org/10.1016/J.LEARNINSTRUC.2007.09.003.

Wahid, Muhammad Iqbal, and Sadia Khan. 2025. “" Two Stars and a Wish": A Feedback Strategy for Reflective Learning in Medical Education: Feedback Strategy in Medical Education.” *Journal of Rehman Medical Institute* 11 (2): 38–40. https://doi.org/10.52442/jrmi.v11i2.953.

Webb, Mary, and Jane Jones. 2009. “Exploring Tensions in Developing Assessment for Learning.” *Assessment in Education: Principles, Policy & Practice* 16 (2): 165–84. https://doi.org/10.1080/09695940903075925.

Whiteside, Aimee L, Amy Garrett Dikkers, and Karen Swan. 2023. *Social Presence in Online Learning: Multiple Perspectives on Practice and Research*. Taylor & Francis.

Wiliam, Dylan. 2011. “What Is Assessment for Learning?” *Studies in Educational Evaluation*, Assessment for Learning, 37 (1): 3–14. https://doi.org/10.1016/j.stueduc.2011.03.001.

Sequential vs. Simultaneous: Approaches to Learning Programming and Statistics

Robert S. Young

Edinburgh Medical School, University of Edinburgh; Zhejiang University - University of Edinburgh (ZJE) Institute
robert.young@ed.ac.uk

Rebecca L. Colquhoun

Department of Earth Science and Engineering, Imperial College London
r.colquhoun23@imperial.ac.uk

Tiago A. Marques

School of Mathematics, University of St Andrews
tiago.marques@st-andrews.ac.uk

Brittany Blankinship

Edinburgh Medical School, University of Edinburgh
B.Blankinship@ed.ac.uk

Ozan Evkaya

School of Mathematics, University of Edinburgh
ozan.evkaya@ed.ac.uk

William P. Kay

School of Biosciences, Cardiff University
kayw@cardiff.ac.uk

Introduction

We live in a data driven world. Much research in STEM departments requires skills in data description, prediction, and inference. These skills are often developed in statistics modules yet increasingly statistical analysis requires computer programming. Therefore, for many undergraduate degree programmes, two closely interrelated but technically distinct disciplines – programming and statistics – must often both be taught. There are different possible approaches to teaching these subjects, most notably whether to deliver them simultaneously or sequentially. In this chapter, we focus on undergraduate level teaching in subject areas that are not primarily programming or statistics. We discuss the advantages and disadvantages to three different approaches and provide guidance on best practice in instructional technique and curriculum design. Case-study examples based on our own teaching experiences are provided throughout to illustrate these strategies.

The topic of teaching statistics to non-specialists has been discussed at length (Kelly 1992; Mustafa 1996; Metz 2008; Gimenez et al. 2013; O'Hara 2016; Bromage et al. 2021). How to do so while acknowledging that programming is an important and related skill has received less focus, which we address here. In all scenarios teachers should be mindful of their own educational context, and, inter alia, what knowledge and skills they want their students to obtain from learning both programming and statistics.

Teaching Statistics Before Programming

This is perhaps the original or "old school" approach. As authors, many of us were taught in this way in our own undergraduate degrees prior to the widespread use of high quality, open-source programming languages (e.g. Python, R). Drawing on our experience, we have found that a key advantage of this approach is its focus on core theoretical concepts in statistics (e.g. probability distributions, uncertainty) without the additional cognitive load associated with technology or implementation. In disciplines where students are likely to need to perform statistical analyses across different contexts and using a range of technologies, this emphasis on foundational statistical reasoning has been particularly valuable.

In practice, we have found that this approach cultivates in students both the ability to conduct 'back-of-the-envelope' analyses in settings where computers are not always available (such as during fieldwork), and enables them to 'think on their feet' when off-the-shelf methods aren't suitable. Importantly, we have observed these behaviours emerge spontaneously rather than through explicit scaffolding.

A second benefit of this approach is that it foregrounds statistics as a discipline in its own right, independent of computation, which can be obscured when programming and statistics are taught together. However, in undergraduate contexts where the primary subject area is not statistics or programming, separating the two can lead to the links between statistics and programming being weakened, potentially reducing insight into the applications of both. This has reinforced for us the benefit of applying statistical understanding once a conceptual foundation has been established. Indeed, if we choose a statistics-first approach then we strive to ensure that students are afforded opportunities to use computational approaches (e.g. programming or

domain-specific interfaces such as STATA) to put their statistical understanding into practice at a later date.

Example

In a first-year geoscience course, statistics was taught independently of programming before moving to a combined approach using Python. Later in the year, during fieldwork, students were asked to measure the size of some ammonite fossils. As they were comfortable with pen-and-paper statistical analysis, students took the initiative to more rigorously characterise the distribution of ammonite sizes, and to estimate the number of observations required to obtain a representative sample. Whilst the statistics course subsequently saw students applying programming to address statistical questions, this in-the-field investigation was facilitated by students first learning to do statistics by hand. Student engagement with 'back-of-the-envelope' statistics in the field emerged autonomously, but as a result we would consider highlighting the potential benefit of this sequencing explicitly for all students.

Teaching Programming Before Statistics

An alternative approach we have adopted in some contexts is to teach programming first, followed by statistics lessons that make explicit use of students' programming skills. Here, students have time to consolidate their programming skills and build their programming mindset. They can then build knowledge and understanding through computational investigation of statistical concepts, before being introduced to formal statistical definitions.

This approach requires more curriculum time than many programmes can accommodate. It also requires that the programming and statistics courses are designed with interaction in mind. For example, using the same programming language in both courses, or intentionally keeping the programming course language-agnostic (e.g. giving examples in multiple languages). We suggest that if this alignment is absent, students may find it more challenging to apply their programming knowledge to statistical contexts.

We have also observed that teaching statistics in a manner which is reliant on the use of programming can present a barrier for students who are less confident in this, particularly early in the curriculum. Indeed, depending on the overall curriculum design, students may have forgotten much of the programming learnt by the time they are learning statistics. We have particularly observed this skill fade at the start of a subsequent academic year following a long summer break. This has led us to conclude that this approach is most effective when programming instruction is closely timed with subsequent statistical teaching, and that particularly between years there is the need to build in time for refresher training. We would avoid a programming-first design when coordination across modules is weak or when a long gap makes knowledge decay likely.

Example

In a second-year course in biomedical informatics, the concept of variance was introduced using a programming-first approach prior to presenting the formal statistical framework of Analysis of Variance (ANOVA). Students were first shown how to use simulations (sampling with replacement) to compare differences between pairs of samples within and across treatment groups. These empirical results were then used to derive the probability of observing a difference between groups and to introduce the concept of variance as a quantitative measure. In the following week, the formal mathematics underlying ANOVA were introduced. Students subsequently learnt how to implement ANOVA using R and, when appropriate, to perform post-hoc pairwise tests to interpret observed differences among treatment groups. We found that this deliberate sequencing helped students to comprehend the underlying theory before learning how to implement the analysis and interpret the output in real-world scenarios. Of course, a trade-off in this approach was the additional contact time needed to run simulations and then later revisit notation, requiring a reduction of coverage elsewhere.

Teaching Programming and Statistics Together

The final approach we have identified is to teach programming and statistics together (Sarvary 2014). In practice, we have seen this implemented in two ways: within a single module, instructors can teach either programming or statistics first and immediately apply it to the other, or the two can be truly interleaved e.g. by introducing new programming constructs and statistical concepts in parallel.

This integrated approach has clear pedagogical advantages. Most importantly, students have reported that they feel able to engage immediately with statistical concepts through computation, and that this can promote a deeper understanding of statistics and its application. Furthermore, this can be supported by introducing new programming approaches when required alongside novel statistical techniques. Students may take a constructivist learning approach to building their statistical knowledge, e.g. by running simulations and adapting code provided in class. For students whose prior negative experience with mathematics presents a barrier to learning statistics (Pletzer et al. 2010), we have found that introducing statistical concepts through code snippets can reduce this anxiety compared to presenting the same idea using formal mathematical notation. That said, our experience also highlights that for some the opposite is true, i.e. that they find it easier to explore concepts programmatically after a formal mathematical introduction.

Despite the benefits of this approach, we have found that the principal challenge of teaching programming and statistics together is the risk of cognitive overload. When both topics are taught concurrently, students are not always able to consolidate new material into their long-term memory and thus the working-memory load is higher (Sweller 2018). In practice, many students will experience hurdles in their learning of both statistics and programming, and these may compound each other to further impede student attainment. We have observed that this effect can be exacerbated for students who are not learning in their native language, as they need to acquire domain-specific vocabulary in two topics simultaneously.

While thoughtful course design - such as carefully and explicitly scaffolding topics, and limiting weekly learning outcomes - can mitigate cognitive overload, it is likely to remain higher when teaching programming and statistics together than with standalone courses (Auker and Barthelmess 2020). Additional trade-offs have also emerged. In some implementations, students' development of broader computational or algorithmic thinking was reduced, since programming was used almost exclusively to perform statistics. For disciplines where programming is primarily used for this purpose, this may be an appropriate outcome. However, where courses aim to set students up to progress to higher-level programming courses, we have found that students benefit from learning to apply programming skills across a broader range of applications to widen their transferable competence. Finally, the choice of programming language and environment can substantially impact learning outcomes, both positively and negatively. This highlights the need for careful course design, e.g. choosing a beginner-friendly programming language when adopting a fully integrated approach. We would therefore recommend avoiding a fully integrated design when class time is too tight, or when support for learners studying in a non-native language is limited.

Example

In a first-year undergraduate course with students from several mathematically oriented degree programmes, concepts such as linear modeling were introduced alongside their computational implementation. During lectures, slides either showed the mathematical and code representations side-by-side or sequentially within the same session, allowing students to make connections across the two representations.

Students then participated in workshops where context-specific exercises with new datasets were given, requiring students to start afresh each time. Students needed to either adapt code snippets based on the given data, or write new code to address each task. We found that regularly changing datasets encouraged transfer of knowledge rather than rote replication. We also learnt that with this style of delivery it was necessary to provide sufficient pauses - either during or between classes - to allow students to fully digest the content from both disciplines, meaning that less time was available for other activities.

Practical Strategies for Teaching Statistics and Programming

Drawing on our teaching experience across a range of programmes, we now summarise a number of practical strategies that we have found effective when teaching programming and statistics, regardless of whether these topics are taught sequentially or simultaneously.

First, we have learnt that embedding the teaching of these topics into the subject matter of the overarching course is critical for student engagement, particularly when programming or statistics are not the principal focus of the degree programme. This embedding may take multiple forms, such as creating appealing visualisations, using authentic datasets, or discussing the conclusions that can be obtained in the students' particular research field. In our experience, this contextualisation helps students understand not only how to perform relevant analyses, but why they are important for their discipline.

Second, statistical methodologies are most effectively taught when also described clearly using a computational approach. For example, while it is helpful to describe the formal mathematical derivation for generating a test statistic, we recommend that this is routinely followed up with the programming approach required to generate the same result. Demonstrating the same concept from both perspectives helps students to consolidate the underlying statistical concepts and to see how statistics are applied in practice, highlighting the close connection between the two approaches.

A third strategy that has emerged from our teaching is to foster a "programming or coding mindset". We found it important to foreground programming as an important transferable skill in its own right, rather than solely being a tool to perform statistical analysis. This involved teaching students how to interact with data programmatically rather than relying on graphical user interfaces or apps alone. In practice, this included providing explicit instruction in code annotation, file paths, naming conventions, and being able to discriminate between characters, numbers, and special symbols. We also found that dedicating time to debugging code – particularly through live programming in lectures – helps to demystify programming and normalise common programming errors.

Finally, our experience strongly suggests that motivating and enthusing students about both statistics and programming is critical. For example, framing computational statistics as a means to increase reproducibility, transparency, and confidence in scientific discoveries helps students appreciate its broader value. We have also found that highlighting the fact that these skills are universally applicable, lucrative, and in high demand in the job market, including beyond academia, is useful for motivating engagement. As an illustrative example, note that the gap in data-driven technical skills reported in 2023 in the UK includes specialist data skills (including statistics), alongside programming and software engineering skills required to manage and analyse datasets (Fearns and Harriss 2023).

Example

In a postgraduate course covering both Python and R, we introduced a recurring activity designed to foster a "coding mindset", called 'Error of the Week'. This feature celebrates errors as learning opportunities rather than sources of embarrassment or frustration. Students were encouraged to share errors on a discussion board, including solutions if applicable. As a course team we responded to every post, and during live sessions the course lead or tutor highlighted a particularly interesting or common error to the class, debugging it live. Over the duration of the course, these discussion boards evolved into a common error glossary for all. We found that these strategies lowered the psychological barrier that programming can often present to students, allowing for more time to be devoted to understanding and interpreting statistical outputs.

Conclusion

We have outlined three approaches for teaching programming and statistics: teaching statistics first; teaching programming first; and teaching both skills together. Our experiences suggest that each approach has clear strengths and limitations, and will be best suited in different settings. In deciding which approach is most suitable it is important to consider the desired learning outcomes, overall programme design, and the cognitive load placed on students, all of which will vary depending on educational context and prior student experience.

Teaching statistics before programming is particularly useful when the priority is to develop strong, transferable statistical reasoning. Teaching programming before statistics can work well when early development of computational confidence is a key goal, and when there is sufficient time and alignment to support knowledge transfer. With careful instructional design, and sufficient curriculum time, many of the affective barriers students experience when learning programming and statistics can be mitigated. Under these conditions, our preferred approach would be to teach programming and statistics together, as this enables students to engage immediately with statistical concepts through computation, and develop a complementary understanding of both. We hope that by highlighting the advantages and disadvantages of different approaches and suggesting practical teaching strategies, this chapter will help educators to make informed decisions about how best to teach and encourage the learning of programming and statistics to their students.

References

Auker, Linda A., and Erika L. Barthelmess. 2020. “Teaching r in the Undergraduate Ecology Classroom: Approaches, Lessons Learned, and Recommendations.” *Ecosphere* 11 (4): e03060. https://doi.org/https://doi.org/10.1002/ecs2.3060.

Bromage, Adrian, Sarah Pierce, Tom Reader, and Lindsey Compton. 2021. “Teaching Statistics to Non-Specialists: Challenges and Strategies for Success.” *Journal of Further and Higher Education* 46 (1): 46–61. https://doi.org/10.1080/0309877X.2021.1879744.

Fearns, Joshua, and Lydia Harriss. 2023. “Data Science Skills in the UK Workforce.” Parliamentary Office of Science; Technology. https://doi.org/10.58248/pn697.

Gimenez, Olivier, Fitsum Abadi, Jean-Yves Barnagaud, Laetitia Blanc, Mathieu Buoro, Sarah Cubaynes, Marine Desprez, et al. 2013. “How Can Quantitative Ecology Be Attractive to Young Scientists? Balancing Computer/Desk Work with Fieldwork.” *Animal Conservation* 16 (2): 134–36. https://doi.org/10.1111/J.1469-1795.2012.00597.X.

Kelly, Martyn. 1992. “Teaching Statistics to Biologists.” *Journal of Biological Education* 26 (3): 200–203. https://doi.org/10.1080/00219266.1992.9655273.

Metz, Anneke M. 2008. “Teaching Statistics in Biology: Using Inquiry-Based Learning to Strengthen Understanding of Statistical Analysis in Biology Laboratory Courses.” *CBE–Life Sciences Education* 7 (3): 317–26. https://doi.org/10.1187/cbe.07-07-0046.

Mustafa, R Yilmaz. 1996. “The Challenge of Teaching Statistics to Non-Specialists.” *Journal of Statistics Education* 4 (1). https://doi.org/10.1080/10691898.1996.11910504.

O’Hara, Robert B. 2016. “On Teaching Ecologists Contemporary Methods in Statistics.” Wiley Online Library. https://doi.org/10.1002/ecy.1325.

Pletzer, Belinda, Guilherme Wood, Korbinian Moeller, Hans-Christoph Nuerk, and Hubert H Kerschbaum. 2010. “Predictors of Performance in a Real-Life Statistics Examination Depend on the Individual Cortisol Profile.” *Biological Psychology* 85 (3): 410–16. https://doi.org/10.1016/j.biopsycho.2010.08.015.

Sarvary, Mark. 2014. “Biostatistics in the Classroom: Teaching Introductory Biology Students How to Use the Statistical Software ‘r’ Effectively.” *Tested Studies for Laboratory Teaching Proceedings of the Association for Biology Laboratory Education* 35 (January): 405–7.

Sweller, John. 2018. “Instructional Design.” In *Encyclopedia of Evolutionary Psychological Science*, 1–5. Springer.

Part II — Teaching Programming in Context: Design and Practice

Serveh Sharifi Far

School of Mathematics and Maxwell Institute for Mathematical Sciences, University of Edinburgh
Serveh.Sharifi@ed.ac.uk

Christopher Aldous Oldnall

Edinburgh Medical School, University of Edinburgh
chris.oldnall@ed.ac.uk

Effective programming education requires thoughtful design that responds to learners and their context. Decisions about technologies, learning activities and collaboration are shaped not only by programming concepts, but also by the characteristics of learners themselves. This is particularly important in contemporary programming education, where learners come from different disciplinary backgrounds, possessing a variety of prior experience. The chapters in this section explore three interconnected aspects of programming education. First, they consider how learning can be strengthened through collaboration, peer interaction and the dynamics of learning within groups. Second, they examine how educators can support learners as they begin their programming journey and develop confidence with programming languages and tools. Finally, they explore how creative and unconventional teaching approaches can broaden participation, increase engagement and encourage learners to think differently about programming and computational problem-solving.

Together, these perspectives demonstrate that programming education extends beyond the choice of programming language or curriculum content, highlighting the interplay between collaboration, learner development, and experimental teaching approaches. The chapters illustrate how active choices of programming tools and environments support the development of confidence and technical competence. From group-based learning and hybrid classrooms to visual, musical, and unplugged approaches to computational thinking, these contributions illustrate the wide range of pedagogical strategies available to programming educators. Collectively, they offer practical insights into designing learning experiences that are responsive to learners, contexts, and educational goals.

Theme — Drawing on Group Dynamics for Programming Education

This theme explores how collaboration, peer interaction, and cohort diversity can be leveraged to enhance programming education. Together, the chapters examine approaches that promote inclusive participation, strengthen engagement, and support learning across a range of educational contexts.

Peer Programming in Action: Pair Programming in Larger Groups (Noè, McManus, and Xu 2026)

This chapter expands on the concept of pair programming by introducing “Peer Programming,” applying collaborative principles to larger groups. With five case studies from a 20-week course, it offers a practical look at the benefits and challenges of implementing peer programming.

This insight is helpful for teachers who aim to boost student engagement through collaborative methods.

Leveraging the Heterogeneity (Michalickova, Holt, and Cooling 2026)

This chapter tackles the complexities of teaching a diverse cohort with varied skill levels and motivations. The authors share a refined framework for addressing these differences, combining quantitative and qualitative analyses to improve teaching techniques.

Educators facing similarly diverse classrooms will find this chapter valuable for its insights and adaptable strategies, particularly those interested in integrating research into teaching practice.

Learning Together Across Modes (Alex, Llewellyn-MacRae, and Orzechowski 2026)

This chapter presents an ambitious approach for integrating onsite and remote students in hybrid teaching. By addressing technological, social, and pedagogical challenges, the authors craft a successful model for fusion teaching, where onsite and remote learners collaborate seamlessly.

This chapter is helpful for educators seeking to push the boundaries of online learning and explore new avenues in hybrid education.

Theme — Teaching Programming to Novices: Python and Beyond

This theme focuses on supporting learners as they develop programming knowledge and confidence. The chapters explore the use of Python for data analysis, the role of tools such as Jupyter in programming education, and the challenges of transitioning to additional programming languages. Together, they provide practical insights into designing learning experiences that support both immediate application and longer-term development.

Teaching Python as a Computational Tool (Sharifi Far, Qu, and King 2026)

This chapter provides a practical guide for integrating Python into an introductory data science course. The chapter addresses the challenge of teaching statistical and programming skills together, offering a step-by-step approach that helps students connect technical coding skills with data science concepts.

Educators looking to balance theory with practical application in data science education will find this chapter useful.

Notebooks for Novices? Pros and Cons of Jupyter (El Gemayel, Budiarto, and Bell 2026)

This chapter dives into the role of Jupyter Notebooks in programming education. The chapter explores the benefits of using Jupyter for teaching core programming concepts without overwhelming students with technical setup issues. It also discusses potential pitfalls and presents ways instructors can optimise the learning experience.

This chapter is a valuable read for those teaching in Python tech-heavy courses.

Bridging Languages: Teaching C to Python Novices (Skipsey et al. 2026)

This chapter tackles the challenges of introducing a second programming language to students already familiar with Python and Jupyter. It focuses on the transition to learning C, highlighting typical difficulties and offering solutions to ease the process.

This chapter is helpful for educators who aim to guide students through diverse programming paradigms, teaching them to adapt and expand their programming toolkit.

Theme — Thinking Outside the Box: Designing Engaging Learning Experiences

This theme focuses on supporting learners as they develop programming knowledge and confidence. The chapters explore the use of Python for data analysis, the role of tools such as Jupyter in programming education, and the challenges of transitioning to additional programming languages. Together, they provide practical insights into designing learning experiences that support both immediate application and longer-term development.

Co-Designing Programming Teaching Experiences (Quinlan and Queiroz 2026)

This chapter draws inspiration from role-playing games and anarchist pedagogies to create dynamic and collaborative learning environments.

By organizing learning activities without traditional hierarchies, it helps educators design courses that foster student agency and teamwork.

Removing Barriers by Programming Without Computers (Cutting 2026)

This chapter tackles programming education barriers by teaching computational thinking without computers. By using games and tangible resources, it provides a low-pressure environment that emphasizes problem-solving skills.

This approach is valuable for diverse groups, including non-traditional learners.

Sound and Music in Programming Pedagogy (Mudd et al. 2026)

This chapter connects sound, music, and programming and how these can be combined in programming teaching. The authors provide a rich set of resources in numerous programming languages that educators can adapt and use in their teaching.

This chapter is for you if you are interested in sound, music, or programming and would be excited to bring these domains together.

Ways to Teach Online (Cooling et al. 2026)

This chapter explores practical approaches to online and blended programming education, drawing on experiences from MOOCs, self-paced learning, and professional education programmes.

It offers insights into course design, learner engagement, assessment, and feedback in digital learning environments.

Seeing Before Coding, Doodling Before Doing (Goopy 2026)

This chapter focuses on a data visualization course that encourages students to sketch and doodle. The approach described in this chapter nurtures critical thinking and creativity, urging students to conceptualize data visually before coding - a crucial skill in an AI-driven world.

This chapter is for educators who are interested in teaching data visualisation as a challenging and creative activity that requires engagement and real-world thinking rather than code output.

References

Alex, Beatrice, Clare Llewellyn-MacRae, and Pawel Orzechowski. 2026. “Learning Together Across Modes: Online and on-Site Pair Programming in a Fusion Course.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Cooling, Chris, Nick Jayanth, Samantha Alvarez-Madrazo, Leila S. Shafti, Joseph El Gemayel, and Lucia Michielin. 2026. “Ways to Teach Online: Lessons Learned Through Experience.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Cutting, David. 2026. “Removing Barriers by Programming Without Computers.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

El Gemayel, Joseph, Arif Budiarto, and William Bell. 2026. “Notebook for Novices? Pros and Cons of Jupyter.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Goopy, Suzanne. 2026. “Seeing Before Coding, Doodling Before Doing: How Teaching Data Visualisation Transforms the Way Students Think.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Michalickova, Katerina, Jonathan Holt, and Chris Cooling. 2026. “Leveraging the Heterogeneity: Teaching Computing Skills to a Multidisciplinary Cohort with a Variable Skill Level.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Mudd, Tom, Charlotte Desvages, Mike Taverne, Matthew Hamilton, and Yashique Chalil. 2026. “Practical Approaches to Using Sound and Music in Programming Pedagogy.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Noè, Umberto, Franziska McManus, and Eileen Y. Xu. 2026. “Peer Programming in Action: Pair Programming in Larger Groups.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Quinlan, Maeve Murphy, and Francisco Oliveira de Queiroz. 2026. “Dungeon

Crawlers and Anarchists: Co-Designing Programming Teaching Experiences." In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Sharifi Far, Serveh, Ruini Qu, and Stuart King. 2026. "A Practical Guide to Teaching Python as a Computational Tool in an Introductory Data Analysis Course." In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Skipsey, Sam, Gordon Stewart, Jeremy Singer, and David Cutting. 2026. "Bridging Languages: Teaching c to Python Novices." In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Peer Programming in Action: Pair Programming in Larger Groups

Umberto Noè

School of Philosophy, Psychology and Language Sciences, University of Edinburgh
Umberto.Noe@ed.ac.uk

Franziska McManus

Edinburgh Medical School, University of Edinburgh
z.mcmanus@ed.ac.uk

Eileen Y. Xu

Division of Psychiatry, Institute for Neuroscience and Cardiovascular Research, University of Edinburgh
eileen.xu@ed.ac.uk

Introductions

Umberto: I am a lecturer in the Department of Psychology at the University of Edinburgh, where I have been teaching data analysis and programming skills for many years now. I am the course organiser of a large first-year course called Data Analysis for Psychology in R1[79], or DAPR1 in short. This is typically taken by first-year Psychology undergraduate students, but the course also welcomes students from other disciplines.

Franziska: I am a Teaching Assistant for the MSc "Data Science for Health and Social Care" at the University of Edinburgh, where I teach R and Python via pair programming. As a tutor, I also teach R and statistics to second-year medics and first-year psychology students. This is my first year tutoring in the DAPR1 course.

Eileen: I am a PhD student based in the Division of Psychiatry at the University of Edinburgh. I have been a tutor since 2022, and have taught R and statistics on undergraduate and masters-level courses in Psychology. This is my third year as a tutor on DAPR1.

Background

"Peer programming" is what I (Umberto) like to call the extension of pair programming principles (Desvages et al. 2026) to groups of more than 2 students. The name highlights how this approach blends the benefits of peer support and pair programming. After multiple years of hands-on teaching experience using this approach, I believe its pros outweigh the cons. This chapter presents five case studies to illustrate peer programming in action, why it was implemented, which challenges it addressed, and the successes encountered.

DAPR1 is a 20-week long in-person course (10 weeks per semester) and a typical week on the course involves:

- Two lectures of one hour each. The first focuses on statistical concepts/methods; the second on how to implement concepts/methods in R
- A one-hour lab. This is a practical session, where students work to solve tasks with the support of tutors
- Assigned reading to consolidate knowledge
- A weekly quiz on the previous week's material to ensure that students keep up with the course as its content is cumulative.

The 20 weeks are organised as four blocks of 5 weeks, where each teaching block focuses on a specific topic: exploratory data analysis, probability, inference, and common hypothesis tests. Figure 18.1 shows a visual schematic of the course. The course materials can be found on GitHub[80] and the lab worksheets are visible here[81].

[79] https://www.drps.ed.ac.uk/25-26/dpt/cxpsyl08013.htm
[80] https://github.com/uoepsy/dapr1
[81] https://uoepsy.github.io/dapr1/2526/labs/

Semester 1

Block 1	Week 1	Formative Report A
	Week 2	
	Week 3	
	Week 4	
	Week 5	
Break		Formative Feedback
Block 2	Week 6	Formative Report B
	Week 7	
	Week 8	
	Week 9	
	Week 10	
Break		Formative Feedback

Semester 2

Block 3	Week 1	Formative Report C
	Week 2	
	Week 3	
	Week 4	
	Week 5	
Break		Formative Feedback
Block 4	Week 6	Assessed Report
	Week 7	
	Week 8	
	Week 9	
	Week 10	
Break		Summative Feedback

Figure 18.1: The structure of the DAPR1 course. It is a year-long course, divided into two semesters of 10 weeks each. The 10 weeks of each semester are further split into blocks, each comprising five weeks of teaching. In the course labs the students produce reports, which they submit in the final week of the block. After each block they receive feedback on how to improve and solutions. The first three reports are formative, the last one is assessed.

When in-person teaching resumed after COVID, student attendance at labs dropped substantially. After a couple of labs we regularly had more tutors than students. I wanted to find a solution to ensure students didn't miss out on important contact time during which they could get support and guidance from staff. When the lab is over, the allocated hours of help are gone, and students have missed out on an important learning opportunity. Going to an empty lab is also demoralising for tutors who are there to help and end up waiting for an hour with no questions from students.

Based on student feedback and teaching experience gathered before I introduced peer programming, I noted three elements were key in driving engagement and attendance.

First, the transition from high school to university is a big change and for many students it can be isolating. Having a place to meet peers which also facilitates the start of new friendships is key to fostering a sense of community and belonging in a new environment. Students specifically left feedback that they didn't feel part of a community and did not have a sense of belonging.

Second, students tend not to attend in-person labs if the worksheets include solutions. They just read through questions/solutions from the comfort of their home and post questions on the discussion forum. I tried withholding the solutions and only releasing them at the end of each week, but many students would choose to work at a one-week lag and on a given week they would work through the previous week's worksheet which had visible solutions, rather than the current week's worksheet.

Third, students left feedback that, as part of their courses/degree, they wanted more practice of writing lab reports to strengthen research skills such as planning, analysing, and disseminating findings. This was raised as part of the British Psychological Association re-accreditation meetings for our degree.

Taken together, the three elements above gave me a sense that it was key for students to work across multiple weeks to create an end-product, as a way to ensure that students engage through a sense of ownership of that product. Furthermore, working as part of a group introduces a sense of responsibility towards their group, so students show up to not let their team down. The steps I implemented, as discussed in this chapter, led to attendance at labs ranging roughly between 75% and 95% for four years in a row. Most importantly, they led to new friendships and support networks students could turn to for help.

Case Study 1: From Pair to Peer Programming

Peer programming was my response to applying pair programming to a large cohort, and it means pair programming in larger groups. DAPR1 has varied in size over the years, but it's roughly been between 210 and 320 students. When planning the number of labs to run each week and the number of student groups to form the following constraints had to be taken into consideration:

1. There is a limit on the number of tutors we can hire
2. There is a limited number of "teaching studio" rooms, i.e. computer labs with a groupwork layout (see Figure 18.2)
3. The teaching studios are heavily booked
4. The teaching studios typically have a small number of tables

The above constraints limit the number of labs we can schedule each week, as well as the number of groups we can form for pair programming within each lab. In the end, I settled for 4 labs each week, and I ask students to work in groups of at most 5 students, where one group occupies a table. I couldn't feasibly ask students to work in pairs due to space constraints and having multiple pairs at the same table was distracting for the groups and not appropriate when the work produced is to be submitted as an assessment.

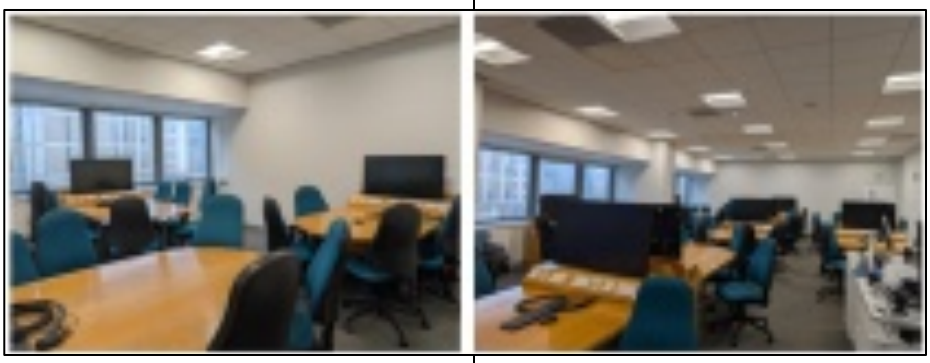


Figure 18.2: Layout of a "teaching studio" room used for the DAPR1 course labs. There are multiple tables in the room, each with 5-6 chairs and a shared PC screen, which facilitates discussions and groupwork.

When students attend their first lab session, they self-select a table as their group, and that will be their group for a semester. As with the standard pair programming model, one person in each group acts as the driver during a given session and the others act as navigators. Here, the driver is the student responsible for writing the code with the keyboard, while the navigators read the task, consult documentation pages, suggest the strategy to follow and fix any typos. Therefore, the driver has the support of multiple peers rather than just one navigator. Typically students self-organise such that one navigator has the tasks to complete open, another focuses on the R documentation/help pages and others suggest code and/or spot typos. The driver rotates after each session, ensuring that everyone in the group has contributed in a coding capacity (as driver) and in a strategic thinking capacity on how to solve a task (as navigators).

Franziska: As a tutor, I check in with students to make sure that they are switching driver each session. To navigators, I offer guidance on how to use the course page and (sometimes) how to access the R help documentation. One aspect of pair programming which usually disappears is the idea that the driver should narrate their experience to give navigators an insight into their aims and thought process. I know this can feel uncomfortable to students, so I only step in when there is a group sitting in silence and the navigators are clearly lost - as sometimes happens when the driver is more confident than other members of the group.

Eileen: As a tutor, I try to be conscious of which students do/don't contribute, and whether they might be having difficulty keeping up. It can be difficult to ask for help on topics you're having trouble understanding, especially in a group setting. As a tutor, I try to alleviate this when I explain things to a group, and make sure that individual students aren't "singled out" for not understanding something. I often start by asking questions about the week's concept/code; group members with clearer understanding are usually the ones to answer, and I can prompt them to lay out their thought process, filling in any gaps. This 1) allows students with clearer understanding to consolidate their knowledge and build confidence by explaining concepts to their peers, 2) gives students who are having difficulties the option to ask for help or pose follow-up questions without singling them out, and 3) facilitates discussion in quieter groups, as students can direct questions and suggestions to a tutor (one person) instead of the whole group. Everyone gets to reap the benefits, and over the semester I get to watch students turn to me less and less as they become more confident in their ability to problem-solve as a group.

Case Study 2: Peer Programming As Scenario-based Tutor Training

As course organiser, I'm also responsible for hiring and training tutors, an important step to ensure the course runs smoothly. Each year, I hire a team of six tutors (who staff four labs each week, with four tutors per lab, and also help with the marking of assignments). It is essential that tutors receive training, for quality assurance and to ensure they feel confident explaining the material and addressing any issues or computing errors during the labs.

I try to maintain a balance of new and experienced tutors each year, to ensure there is continuity of expertise but also the building of new expertise for the future running of the course. Experienced tutors also act as informal buddies to new tutors who can shadow them in the first few weeks of the course to see hands-on how tutoring works.

Each week, I run a one-hour tutor preparation meeting where we all work through the lab worksheet that students will complete in that week's labs. If students are expected to use peer programming in the labs, I believe that the best way to prepare tutors is to also run the preparation meetings as peer programming sessions. This leads to scenario-based training sessions that allow tutors to get hands-on experience with how peer programming works in practice, as well as the natural issues that appear when making typical typos/errors when typing the code for that week's lab. This ensures the tutors have a preview of typical R errors they might have to help address in the labs.

Since the introduction of peer programming, the tutor preparation meetings have become an interesting discussion-based hour, where different perspectives are heard, new tutors strengthen their understanding, and it's also a safe space to ask questions and seek clarifications as I act as navigator/driver when it's my turn! When I'm the driver I also tend to assume the persona of a novice programmer who needs more guidance.

Franziska: The tutor sessions are enjoyable and create time for tutors to build community and share teaching experiences. They also provide direct access to the course organiser, who can answer questions about the course.

In the tutor sessions, we work through the lab worksheet exactly as the students will, which gives a clear sense of what we are teaching in the upcoming lab. In my experience, these group sessions are quite unusual for tutor training. In other courses, I use allocated time for independent study to prioritise going beyond the lab content, focussing on aspects I understand less well and the wider context of the course (which is sometimes missing from the tutor sessions). However, in a busy schedule, the tutor sessions create protected time to prepare each week. If a course organiser were concerned about their tutors preparing to a minimum standard each tutorial, these tutor sessions would be the safer option.

Alternative approach: In another course I teach on, tutors write a short reflection after each tutorial - an alternative way to flag issues and for course organisers to keep track of how well the course is running. However, this can lead to course organisers fixing issues in retrospect, i.e. after the labs have run, while the tutor session approach is more pre-emptive, allowing for fixes before the labs start.

When it comes to driving, we often take a mixed approach - sometimes tutors pretend to be a novice student, other times we code using our own knowledge as tutors. This stems from the dual goals of the session: to get our coding up to speed and to expose us to questions students may have. At times, the friction created from struggling through each missing space, bad variable name and mistyped pipe can be frustrating (although I know that not all tutors share this view). One group member intentionally misunderstanding the other also hides one of the major positives of pair programming: working together as a team. For tutors who are experiencing pair programming for the first time, I feel it is important to model pair programming in its best light.

The principal aim of the tutor sessions is to learn how to write the code we are about to teach. Although I am confident in my coding abilities, I often feel shy in the peer programming sessions, and am reluctant to volunteer as driver or to offer suggestions as navigator. This comes from a feeling that, as a tutor, I should be an expert on the content already - it is what I was hired for, after all. There is also something about being in a group setting (course organiser included), which feels more vulnerable than it would with a partner.

Alternative approach: I would be curious to try the tutor sessions without a course organiser at the helm (e.g. if they didn't have time or wanted to let tutors work through the code themselves). We would lose the course organiser's insights of what students find difficult, but the experienced tutors would get a chance to fill the gap.

Eileen: As an experienced tutor on the course, I use these preparation meetings to highlight common errors that students might make when completing the week's lab exercises. The majority of DAPR1 students begin the course with no coding or R experience, whereas we DAPR1 tutors enter with more experience. This difference in experience is reflected in the errors we make in R and our subsequent approach to troubleshooting - it can be easy to miss the kinds of mistakes that novice programmers make (e.g. missing parentheses, commas, quotation marks) and jump straight to scouring Stack Overflow. As medics like to put it, "When you hear hoofbeats, think horses, not zebras" - especially during the first few weeks of DAPR1.

I find that modelling peer programming during these early preparation meetings is a useful

way of practicing the kind of troubleshooting we will be doing. At the same time however, the dynamics within the tutor preparation meetings can be very different to the group dynamics seen during class - though this is not unique to the peer programming approach. This can make it more challenging to implement peer programming methods in certain lab scenarios, such as in groups where there are significant language barriers.

Case Study 3: Programming Towards an End-Product

To address the fact that lab worksheets with solutions could be completed from the comfort of home, the lab worksheets corresponding to the same block (each block comprises 5 weeks of teaching) are grouped in such a way that they together lead to the creation of a report. In other words, they are cumulative and each one builds on the context from the previous one.

The first three reports are formative, giving students the chance to obtain feedback when it matters most, i.e. while they are learning rather than at the end of the course. The fourth and last report is assessed. Students submit their work before they receive full solutions of each report and feedback on how to improve. Therefore, the lab worksheets only have tasks and hints available.

Working across multiple weeks motivates students to show up, as the different weeks are intertwined, and there is a shared understanding that people need to turn up and help to avoid the group falling behind, so students are keen to participate. This gives the group a sense of ownership and they tend to care about their work. Asking students to submit their end-product is also key to their engagement and attendance in labs.

Case Study 4: Peer Programming To Foster Belonging

Year 1 of an undergraduate or postgraduate degree is typically a year where students need to form new connections. Peer programming (as well as the standard pair programming) has been a key element to ensure students form a peer-to-peer support network and form new friendships. The student feedback from the DAPR1 course with peer programming implemented speaks for itself:

"I enjoy working in lab groups because I feel like it helps to have other people to learn and figure things out with." - Student 1

"To be honest, I really like the group work of DAPR1. Before, I didn't participate in too many group activities, I prefer to study by myself. But the activities in the lab gave me a deeper understanding of group activities. Although I don't want to admit that I actually have a lot of things I don't understand about the study of DAPR1, the other members of the group helped me make up for it. Compared with self-study, group activities can gather everyone's wisdom, complete homework better, and become good friends with group members. For me, who came to study abroad alone, this is really lucky." - Student 2

"The group work is very helpful. Not only have I made new friends, but I now also have people that I can rely on when I need help. Group work has also made the individual workload seem much lighter and we get things done faster." - Student 3

"Yes I have found it extremely helpful. During week one, I felt extremely lost and confused but my group helped me with that and so I think having that was very beneficial. In addition to this, it helped as different people were better at different aspects than others and it also meant we could spot each other's mistakes

quicker, before R tells you it will not work! I also think that it is really beneficial because it gives you the perfect opportunity to work with other people and make potential new friends." - Student 4

The above feedback, combined with the much improved attendance at labs, highlights the positive effect of peer programming on the student experience. Furthermore, many DAPR1 groups also become "study groups" for other courses and they end up meeting outside of class time.

Eileen: This is a real highlight of DAPR1 for me, as when peer programming works, it can work extremely well. It's a privilege to watch groups of students who start as strangers (who are all understandably disgruntled at having to learn statistics and R) become fast friends by the end of the year, and I think peer programming facilitates this. Making mistakes and asking for help is often scary, but the shared responsibility of peer programming means that errors are never one person's fault - and successes can also be shared as a group. Students who stay in the same groups get to make progress together, and since tutors cover the same groups week-to-week, I also get to celebrate how far they have collectively come since week 1.

Franziska: From my experience of tutoring different cohorts, it seems to me that psychology first-years are particularly amenable to peer programming as a concept. For the most part, they enjoy spending time together (groups often chat while settling in) and navigate social settings with ease - greeting tutors and fellow students alike. Students patiently guide each other through difficulties, so even when the driver is lost, they have supportive navigators to show the way.

Other cohorts may not share the same values. For example, in a course with a high workload, students may need to be highly strategic in their learning to keep up with the demands of the course. If pair (or peer) programming is seen to slow them down, it may be met with resistance. I have seen first-hand how this pressure influences how pair programming as a concept is received. Similarly, if students are interested in "getting the right answer" and don't expect to use programming again, they are less likely to see the value in exploring alternative approaches with other students. In contrast, the psychology students of DAPR1 have surprised me in how often they come willing to learn, even when other assessments loom. I think a large part of this is the welcoming atmosphere in labs and the reduced need for independent study outside of scheduled hours - if you attend lectures and labs, you will learn statistical theory, see it performed in R, and apply it yourself in an assessed (or formative) report. For our first-year psychologists, there is also value in learning things well. They have two years of R courses to go, and a dissertation year after that. It is worth applying good practices and investing for the future.

Case Study 5: Peer Assessment To Encourage Active Contribution and Sense of Ownership

As educators, we may be concerned (and many students are) with the fact that any type of groupwork could have free riders, i.e. people who benefit from the work of others without meaningfully contributing to it. To help address this challenge I wanted to have in place a strategy to reward students who contributed to the end-product, where the "contribution amount" is based on ratings by their peers. With the help of the University's learning technology team, I based the implementation on the WebPA tool by Loughborough University[82].

The assessed report created by each group receives a grade from the marker. 80% of the grade is fixed, while 20% of it is variable. Let me give an example: if a marker assigns a grade of 65 to the group report, 52 is the minimum grade that any group member can receive, and 13 points are distributed based on how much each person contributed as rated by their peers, using a Likert scale where 0 = None, 1 = Very poor, 2 = Minimal, 3 = Average, 4 = Good, 5 = Excellent. The rating is done within WebPA, where students use a form to anonymously rate their group members according to the following criteria:

- Overall, rate the positive impact of each group member's contributions to the project
- Rate the extent to which each group member actively participated in group discussions/meetings
- Rate the extent to which each group member contributed to the analysis strategy

[82] https://webpaproject.lboro.ac.uk

- Rate the extent to which each group member contributed to the code

- Rate the extent to which each group member contributed to the writing of the report

I ask students to self-enrol in groups on Blackboard Learn, the virtual learning platform used by our university. WebPA easily integrates with many virtual learning environment platforms, making it easy to perform the peer assessment of contribution for the same groups created in the virtual learning environment.

The fact that students know that their grades can go up or down based on their contribution has been a key motivator for students to show up and engage with the projects. We did notice more discussions during the assessed report, in comparison to the formative reports which are not peer-adjusted.

The 80-20 percentages were chosen keeping in mind that the tool is not foolproof. First, it can be played if everyone in the group agrees to rate someone in a specific direction. Second, it can happen that a submission which received a passing grade can become a fail after the peer adjustment. The 20% adjustment limits the occurrence of this issue, and a higher level can lead to more of such cases, which are often questioned during scrutiny or moderation.

Discussion

Peer programming has been a welcome addition to the DAPR1 course and, I believe, to the Psychology degree. Students have consistently given good ratings and feedback to the course, and the teaching is very interactive. It has been a pleasure to always enter into a lab room where there are active and lively discussions, and ideas bounce back and forth between students and tutors. Most importantly, students end up forming strong connections and leave the course having established a peer-support network they can turn to for help.

References

Desvages, Charlotte, Brittany Blankinship, Umberto Noè, and Pawel Orzechowski. 2026. “Structured Group Work with Assigned Asymmetrical Roles and Switching: Lessons from Pair Programming Across Disciplines.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Leveraging the Heterogeneity: Teaching Computing Skills to a Multidisciplinary Cohort With a Variable Skill Level

Katerina Michalickova

Early Career Researcher Institute, Imperial College London
k.michalickova@imperial.ac.uk

Jonathan Holt

Mechanical Engineering, Imperial College London
j.holt23@imperial.ac.uk

Chris Cooling

Early Career Researcher Institute, Imperial College London
c.cooling10@imperial.ac.uk

Overarching Themes

To an educator, this could be the stuff of nightmares. A classroom full of bright university students who have nothing in common - discipline, motivation or prior skill level. Their expectations vary, and they all expect you to help them make sense of computing for their future research. Regardless of their differences, all students chose your elective module.

This is precisely the context of a multidisciplinary and multi-skilled audience that our team of educators has committed to addressing since 2019. We designed an undergraduate module and subsequently a framework to navigate this delightful nightmare every spring term for the past five years, gradually refining our approach.

To make this chapter as relevant to programming educators as possible, we avoid fine details of our implementation and define a general framework that allows any educator to facilitate learning for a mixed-interest audience with varying skill levels and motivations. As a secondary effect, this chapter and the framework are relevant to other practitioners interested in student group project work dynamics and in integrating research into teaching practice.

Context: Elective Undergraduate Module Including a Brief History

We describe how we approached the design and delivery of a second-year undergraduate module in *Interdisciplinary Research Computing (IRC)*. The goal of the new module was to introduce our students to programming and related skills relevant to STEMM research and build a better understanding of their application. The plan was to use a variety of pedagogical approaches to motivate and upskill students with variable disciplinary interests and prior programming experience. This included, for example, supporting students' transferable skills and designing an assessment independent of incoming students' programming skill levels.

The IRC module follows the usual UK university format and runs for ten weeks; it is FHEQ level 6 and worth five ECTs. The format comprises 20 hours of face-to-face time evenly distributed over the term, and independent work and study. Our students come from various departments, but mainly from Mathematics, Physics, Life Sciences and Medicine. The module is an elective, and the mark does not appear on transcripts, which may downplay its importance compared to the departmental content. So far, we have been working with a cohort of 28 split into seven groups of four - a fairly standard size for the institutional portfolio of undergraduate elective modules. The teaching team consists of teaching fellows, research fellows, and graduate teaching assistants. Each team member contributes to teaching, marking and group supervision. The module lead coordinates the work, manages logistics, oversees iterative improvements, and supports the GTAs.

So far, we have completed five iterations of the module. This chapter describes the current state of the design, which was initiated in 2019 and underwent yearly improvements based on our observations and student feedback. The overall content and layout have not changed much, the majority of changes concerned supporting group work and introducing a new assessment type.

Contribution: Deep Dive Into the Student Reflections

In this chapter, we go a step beyond a case study. We present a mature design, analyse students' reflections on the module, and conclude with a general framework useful for educators in a similar teaching context.

Our design is the result of collaboration with a large teaching team, informed by systematic analysis of each module iteration and students' feedback. We describe the design in some detail and would like to bring the reader's attention specifically to the Connect 4 individual programming assignment. The module is designed to support individual progress, which makes it difficult to devise a single way to test their programming skills after completing their chosen programming topic. The Connect 4 assessment is our answer to this problem, which has been successfully used for several module iterations.

The last assessment of the IRC module is a reflection in which students discuss their experiences with the module, group work, and how their understanding of research computing has changed. We used reflections from the last two iterations (n=54) to conduct a thematic analysis and systematically identified key themes shaping the module experience. While thematic analysis effectively describes the cohort as a whole, we also examined individual experiences. This was prompted by reflective statements indicating the presence of student experience subgroups. We used a combined approach to explore this topic. This involved comparing self-reported prior skills against final grades and individual grades. We examined an outlier group and cross-referenced it with individual reflections to identify additional factors that shape the overall experience and ultimately determine the quality of learning.

The analysis significantly deepened our understanding of students' experiences. It helped to justify our past improvements and, most importantly, informed our future optimisations. To make this chapter as relevant as possible, we synthesised our experience to date and combined it with the analysis's findings to create a general framework for working with a multidisciplinary, multiskilled audience.

Framing: Rationale and Aims

Through the module, we take students on a learning journey that involves collaborative project work. Student groups design and implement an open-ended project. The activity is supported by a dedicated staff advisor who acts as a coach and an occasional technical expert. The planning stage takes about three weeks, followed by two weeks of writing and editing the project plan based on the advisor's formative feedback. The implementation is done in the second half of the module. Projects are presented at a poster session in the last week of the term.

We give the students considerable freedom to plan their projects, while mitigating the open-ended nature of the work through supervision and structured opportunities for reflection on progress. At the same time, we aim to maximise support for individual learning, making it relevant to the individual's needs. The goal is not simply to teach programming but to create a learning environment that mimics a real research process and fosters students' agency to respond to various learning situations.

A typical example of a project would be a machine learning experiment in which, after researching the topic and data sets, the students would conduct one or more ML experiments on a chosen data set and compare the outcomes with available data in the literature. While the description is short, the reader can appreciate that executing this project involves significant research, planning, programming, coordination and time management.

Our approach explicitly values independence. Many second-year students, however, are still adjusting to university-style learning. We have observed variable readiness for independent work and variable motivation that reflect the high demands of the regular coursework outside the elective. For many, this module is their first opportunity to work in a multidisciplinary setting. Therefore, we also need to address the variable readiness of our students. Over the years, we came to value a coaching approach to group supervision.

The module's design is based on multiple theoretical frameworks. While project-based learning principles (Thomas 2000) are central, interdisciplinary group work, facilitated by collaborative learning rooted in social constructivism (Daniels, Cole, and Wertsch 2007; Piaget 1970; Vygotsky 1978) is the primary dynamic shaping students' experiences. The group work aspect was somewhat underappreciated in the initial design, and consequently, most of the design changes we have introduced so far have aimed at facilitating team dynamics and fostering various transferable collaborative skills.

The design is based on four aims, which address the rationale above:

1. **Enable project-centred group work in interdisciplinary teams:** The central activity of the module is project work done in a multidisciplinary group setting. The group leverages unique student experiences, promotes knowledge sharing, and enables the development of transferable skills.
2. **Simulate the research environment:** We aim to simulate it as closely as possible. In addition to working on a project as part of a multidisciplinary team, students have dedicated advisors, attend guest lectures, discuss their project with visiting scientists, and present the project outcomes in a poster session.

3. **Enable independent learning:** Independent learning happens via online content and group projects (GW label in Table 1). We intentionally design the groups to be as diverse as possible in terms of discipline and prior skills, based on surveys conducted before the module begins.
4. **Carefully scope instruction time:** For the instruction part (LS and HO labels in Table 1), we carefully chose a key topics that provide students with a mental map of the research computing landscape.

The next section describes the implementation of our aims and the design of an appropriate assessment.

Design: Outcomes, Learning Experience, and Assessment

During the module design phase, we defined the following intended learning outcomes to constitute the basis of our students' experience. The outcomes are broad and do not address specific technical learning points. Instead, they create room for every student to have a unique experience connected to their own needs and interests.

1. Reflect on the relevance and explain the importance of research computing and data science in contemporary research.
2. Reformulate a real-world problem as a research question and break it down into achievable steps that can be resolved using programming and research computing tools.
3. Apply programming and research computing tools to solve computational problems and apply principles of collaborative software development.
4. Effectively communicate project progress and results to a multidisciplinary audience.

Table 1 below presents the module content in a nutshell. It is colour-coded based on the type of pedagogical approach used to deliver a particular module part.

Whole-cohort "Hands On" (HO) yellow

"Group Work" (GW) blue

"Lecture Style" (LS) green

"Administrative" (ADM) white.

Each face-to-face session provides a good mix of activities and most of the module involves active learning. The main component is group work on projects of students' own choice (GW). Individual learning takes place via pragmatic instructions (lecture style (LS)) on the fundamental topics and Python self-study (hands on tasks (HO)).

week	**Classroom time:** **5mins-30mins**	**30mins-55mins**	**65mins-90mins**	**90mins-115mins**
0	Skills Survey and Self-Assessment (ADM)			
1	Introduction, Module Information (ADM)	Guest Talk (LS)	Group Introductions and Contract (GW)	Meet your Disciplinary Colleagues (HO)
2	Python and Group Work Q&A (HO)	Final Project Introduction, Guidelines for Project Selection (GW)	Group Work - Identify Commonalities Between Disciplinary Examples (GW)	Guest Talk (LS)
3	Python and Group Work Q&A (HO)	Project Discussion (GW)	Guest Talk (LS)	Introduction to Git (HO)
4	Project Plan Discussion, Group Check-in (GW)	Python and Group Work Q&A (HO)	Introduction to GitHub (HO)	
5	Project Plan Discussion (GW)	Connect 4 Tournament (HO)	Connect 4 Tournament (HO)	Gentle Introduction to Computer Architecture 1 (LS)
6	Gentle Introduction to Computer Architecture 2 (LS)	Visiting Scientists (GW)	Project discussion, Group Check-in (GW)	Finalise Git and GitHub Project Setup (GW)
7	Visualising Data in Python (HO)		Visualising Data Practice (HO)	Group Work with Advisors (GW)
8	Introduction to the Command Line and Command Line Practice (HO)		Group Work with Advisors, Group Check-in (GW)	
9	Introduction to High Performance Computing (LS)		Group Work with Advisors (GW)	
10	Poster Presentations (GW)			Prizes and feedback (LS)

One of the most challenging aspects of the module was creating a fair and effective assessment, considering that students start with varying skill levels. Project work and group assessment can be somewhat unreliable as a measure of students' individual progress (Almond 2009). For this module, we chose a combination of group and individual assessments. Students submit five assessments, three individual and two group ones (45% and 55%, respectively). In the individual assessments, we ask students to reflect and experiment with coding in low-risk environment. The group assessments consist of project plan followed by project presentation via poster and GitHub repo.

Group Work and Group Assessment: From Forming to Storming

We survey students about their coding experience before the module and create interdisciplinary groups with mix of skill levels. Groups are introduced during the first week to maximise time for group forming. Each group has an assigned member of the teaching team with a role of an advisor who is available to them during the face-to-face sessions as well as for any other group meetings.

The initial meeting is structured to facilitate group formation. After introductions and an icebreaker, students are encouraged to collaboratively develop a group contract and establish regular meeting times outside the face-to-face module sessions. We encourage the students to start brainstorming early, so they have ample time during the first half of the module to refine the project plan. We advise them to identify a project that plays to their collective strengths and carefully consider who will take on which task.

A significant portion of time is spent on developing a project proposal with background research, clear milestones including work assignment, defined minimal working product versus extensions, and plans for knowledge sharing. The work does not start until a detailed plan is in place. While we have encouraged individual approach of staff members to group advisory role in the past, we have moved toward stronger coordination across the teaching team and using coaching-like methods.

We group assess an open-ended project proposal and a project presentation via a poster and collaboration documented in a GitHub repository. These assessments emphasise collaborative work, the ability to iteratively plan and improve a project, and deliver a minimal working product. The teaching team reserves the right to mitigate the marks when the group members are not contributing evenly.

The teaching team regularly discusses and monitors the group's progress. These discussions have become a central part of the weekly planning sessions, providing an opportunity for the staff to share experiences with our teaching assistants, who tend to change over the years.

Supporting and Assessing Individual Skills Development: Options and a Low-Risk Environment

Since students join with variable programming skills, we give them a choice from several topics that they commit to study during the first three weeks of the course, with a time commitment of 3-4 hours a week. This enables them to progress their knowledge in the direction that they find useful. Our teaching team has developed a publicly available suite of Python courses with numerous hands-on exercises that are suitable for self-study: *Introduction to Python for Researchers, Intermediate General Python, Object-Oriented Python, Numerical Computing with Python, Plotting with Python, and Data Processing with Python* (ECRI 2026). Self-study is supported by a dedicated channel, weekly Q&A sessions, and quizzes. Students report on their progress in the third week of the module via a short written reflection. Here they demonstrate their progress, and practise for the final course reflection.

Designing a motivating and fair approach to develop and assess individual programming skills in our heterogeneous cohort took us some time. Initially, we avoided individual programming assignments altogether. For the last four module iterations, we have been using a creative design based on the Connect 4 game. Students are given a GitHub repository with a simple, modular implementation of the Connect 4 game. Their task is to study the design and figure out how it works. Next, they edit a function which defines a strategy by generating the next move based on the current state of the board. This function already works, but uses a poor strategy. Students' new code can be immediately tested against a provided random strategy opponent function.

Once the student has developed their code, their mark can be generated repeatedly by automatically playing the student's strategy against this random strategy many times and recording how many games it wins. Once a student figures out how to set up the code, how it works and how to test it, they have already enhanced their learning, especially if they are new to Python. The marking is weighted so students will pass the assignment with a simple strategy. Students who are interested in developing a more sophisticated strategy can spend more time coding and achieving higher marks. Since the assignment counts for 10% of the mark, the advantage of more experienced coders is not disproportionate. We celebrate students' achievements with a Connect 4 tournament between the students' strategies, with an anonymous participation option.

As the last task of the entire module, each student completes a two-page reflection on the module experience and their personal growth. These reflections close the learning loop for individual students and provide a rich source of data for further analysis.

Learning From Students' Reflections

As indicated, the final student reflections on the module serve a two-fold purpose. They help students organise their thoughts on the experience, reason about their contributions, and reflect on what went well and what did not. For the teaching staff, the reflections present a good opportunity to learn about the effectiveness of our design and suggest improvements. For this chapter, we performed a qualitative and quantitative analysis on a set of 54 reflections from the last two iterations of the module (to prioritise the most evolved design). We aimed to 1) draw general conclusions about the module and identify topics that generated the most engagement, and 2) identify if any groups of students do not experience the module as intended and may need a particular adaptation or support.

Thematic Analysis: The Cohort Voice

To frame the reflective assignment, students received instructions on the reflective format and a list of topics they may want to discuss. Some topics were frequently discussed, some hardly mentioned, and some reflections were on topics not present on the list. Students were assured about the use of their reflections to assess the format and not the content of their reflections to encourage honest disclosure. Overall, most reflections subjectively read as broadly neutral, with 9% coming across as mostly negative and 20% as mostly positive. The overall descriptive content of reflections was high, indicating that, for many students, this was the first time they engaged in a reflective exercise. Using a classical thematic analysis approach (Braun and Clarke 2006), we pinpointed three dominant themes that characterise the students' module experience: 1) general opinions on the module as a whole, 2) group work dynamics, and 3) suggestions for future cohorts. Table 2 summarises three main themes and sub-themes.

For the first theme, students appreciated the module. The word "challenge" appeared often, followed by an expression of satisfaction from learning during the process. The module overall is perceived as challenging overall, however students' favourite components were the most complex ones - guest talks, coding assessment, or high performance computing. These components but one are generally not part of the assessment and could become victims of strategic learning. We are happy to report that we found no evidence of this. We were also satisfied to find many unprompted statements indicating the students appreciated the usefulness of the module - either for their respective degree course content, future professional progression or simply as acquiring transferable skills. Interestingly, we observed that only about half of the students chose to elaborate on the evolution of their understanding of computing's place in the research space. There were a few negative comments that indicated a mismatch between expectations from the module and the module content. Most of these students also acknowledged that, after completing the module, they understood the design intent.

For the second theme, the most frequent statements in the reflections concerned group work and dynamics. Group work became a major part of our students' experience, deemed useful and rewarding, yet also difficult g to the point of frustration. We consider this the most important outcome of this analysis. It was revealing that, through the module iterations and reflections, we observed numerous other factors contributing to the heterogeneity of the groups. We have also noted a strong signal from students more experienced and motivated than others: they were frustrated by their peers' inability to contribute on the same level, yet they were also frustrated by not being able to engage them.

For the third theme, students' suggestions for future cohorts were in line with what we recommend and tried to promote during the module. For example, they highlighted a common issue that we have been trying to mitigate - work is often done later than would be optimal. Students know it and wish they did not have to rush at the end. We will use these recommendations directly in our teaching.

Table 2 (next page): Summary of the main themes identified in student reflections. Each theme is divided into several sub-themes, and short explanations and representative quotes are included. Average frequencies per reflection for themes and sub-themes are also included.

Theme / *Subtheme* *(avg. freq.)*	**Representative topics and notes**	**Representative quote**
Opinions on the module as a whole (4)		
General appraisal (1)	Positive or very positive, often using word challenge, a few unmet expectations statements that acknowledge the understanding of intent behind the design.	*"Much appreciation to the absolutely encouraging and skilful teaching staff, and this one-term study is one of my highlights in my university career!" (2023-2024)*
Highlights (1)	Overall, students highlighted challenging topics as most appreciated or interesting - talks by guest scientists, followed by Connect 4 assignment, GitHub and high performance computing	*"One of the most valuable skills that I learnt during the project was learning how to use the collaborative features on GitHub." (2023-2024)*
Usefulness (1.3)	Unprompted (and frequently), students talked about the usefulness of the module - for degree course, future professional progression or for acquiring transferable skills.	*"This experience allowed me to grow both technically and personally, as I navigated the challenges of working in a diverse group and tackled complex problems that required creative problem-solving." (2024-2025)*
Evolution of understanding of research computing (0.7)	Only half of students chose to elaborate. Good understanding - research computing is cross-cutting and requires much more than developing programming skills.	*"This module changed the way I see computing. I used to think of research and computing as two separate things, but now I realise that computing can be a powerful research tool in nearly every field." (2024-2025)*
Group work (4.5)		
General appraisal (2.7)	Group work was commented on most frequently and prompted frequent honest disclosure. A mix of positive statements and expressions of frustration. Multiple factors influencing group work were mentioned: expectation, scheduling and degree course load, motivation, self-efficacy, ethic, health (physical and mental). Challenges: work division causing some students to miss out on learning opportunities, not speaking early enough when group dynamics was suboptimal.	*"I cannot overstate the value of effective communication within our team. Despite our diverse academic backgrounds, we managed to find a common language through the shared goal of the project." (2023-2024)*
Self-appraisal (1.3)	Balanced mix of positive and self-critical. Positive: good progress in technical and professional skills. Negative: lack of confidence (often coupled with less prior experience). Statements that need attention from the teaching team: (1) willing to learn but not finding a way of contributing to the project meaningfully - being an observer, (2) the experienced students completing disproportionate amount of work due to their self-reported inability to involve others effectively.	*"I recognised a need for greater confidence and critical engagement, especially when faced with uncertain methodologies proposed by my team" (2023-2024)*
Appraisal of others (0.5)	Most comments focused on the work ethics of others and came from students who shouldered more work. Very few positive statements about individual colleagues (positive statements generally refer to the whole group)	*"As my group members were inexperienced with coding, whereas I felt more confident, it quickly became clear that I would have to take leadership of the project. I was happy to do this." (2023-2024)*
Message for future cohorts - "I wish I did this..." (1.2)		
Points for group work (0.6)	The statements clustered into the following categories (by order of frequency): select a motivating project for all group members, manage your time better and avoid rush at the module end, be strict about accountability, realistic scoping, divide work fairly and strategically, communicate efficiently and honestly, make an effort to get to know your colleagues.	*"Maintaining strict goals from the beginning and conducting regular meetings so that the project remains on course." (2024-2025)*
Points for personal actions (0.6)	A highly mixed group of statements that show good understanding of what is needed to successfully complete the module: be proactive, don't delay work, learn some Python before the module, ask questions, learn from your peers, realise that everyone has something to contribute, realise the strengths of collaborative work, trust yourself, have open and curious mind, and keep good notes and log of activities.	*"You are studying and working with amazing teammates, please embrace the learning process with an open mind and a curious heart. Don't be afraid to ask questions, seek help from your peers and instructors, and step out of your comfort zone" (2023-2024)*

Student-Centred Analysis: Finding the Essential Details

Qualitative thematic analysis does not give insight into subgroups of students. Thorough analysis of reflections coupled with group membership, prior skill and achievement enables us to identify and focus on supporting groups of students who may find the design not productive.

In order to better understand the diversity of the cohort, we looked at overall module grades grouped by self-appraised programming experience and their background, as shown in Figure 19.1. There is a uniformity of grades that is broadly independent of programming experience and disciplinary background. This was a goal of the assessment design, which prioritises group work and non-technical reflections.

A clearer picture of the students is apparent when we focus on the Connect 4 assignment, which was the only individual programming assessment of the module. Figure 19.2 shows grades grouped by students' self-appraised programming experience and their discipline. Broadly, we see that students from life sciences and medicine came into the course with lower programming experience and overall attained lower marks than their STEM peers. The fact that students with no prior programming experience gained good grades raised concerns regarding the use of artificial intelligence. In an anonymous survey regarding Generative AI use, all students who used AI for this assignment claimed to understand what was being suggested and provided guidance to the AI agent. This confounding factor led us to focus on students who achieved low marks for further analysis.

Figure 19.1: Graph showing the relationship between pre-module self-appraisal and overall module grade.

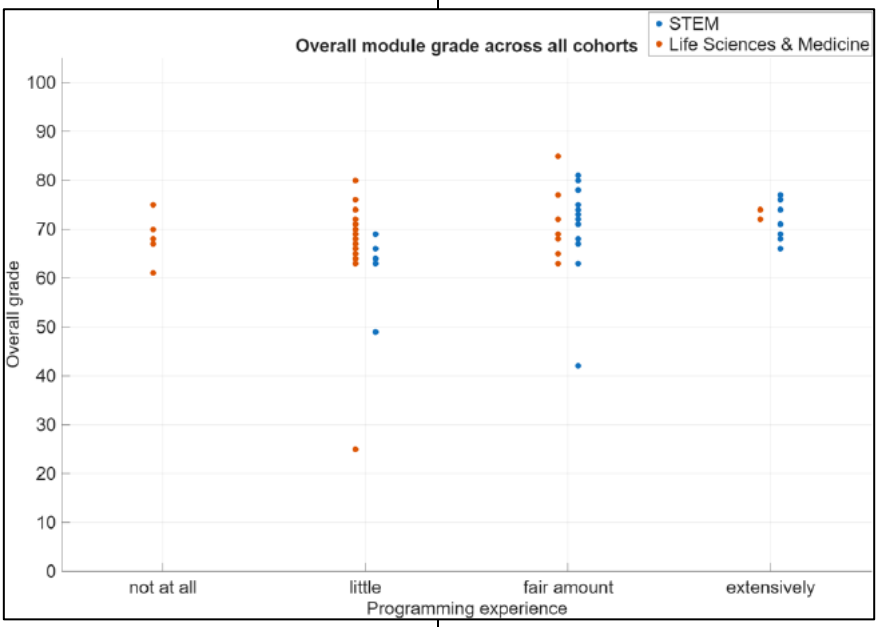
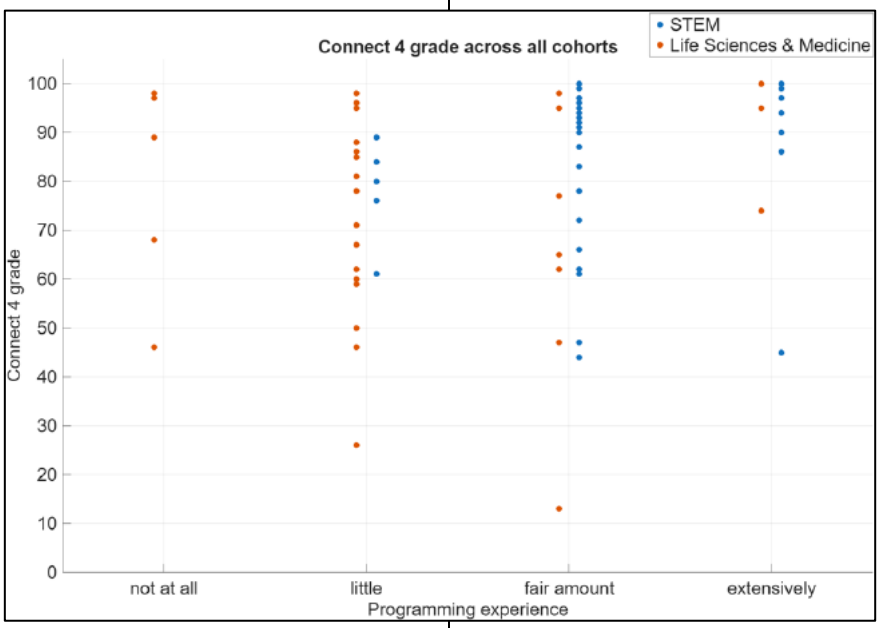


Figure 19.2: Graph showing the relationship between pre-module self-appraisal and Connect 4 programming assignment grades.



We took all students whose Connect 4 grades were lower than 60% and attempted to identify commonalities in their experience in the reflections, focusing on comments about their own performance and the project. The assumption was that this group of students was most likely to run into difficulties and not be able to contribute to the programming aspect of the project.

Project Design Determines Possibilities For Contribution

Out of the 12 students in this group, nine were from Life Sciences or Medicine. Seven mentioned they were not skilled enough to contribute to the programming aspect of the project and ultimately took (or were given) the role of poster designer or researcher. In fact, across the entire cohort, the least experienced students tend to be relegated to the tasks with the lowest friction and, as a result, the lowest learning potential. Even students with self-reported extensive programming experience, but who performed considerably worse than their teammates in the Connect 4 assignment seemed to be given these tasks.

One student, who took a leadership role in their team, reflected that they had been *"very stubborn"* to seek help from their teammates because they knew they *"would not be able to lead as much in the actual coding"*. Stepping back to the most comfortable role led many students to an underexposure to programming, and, ultimately, some regret. The same student continues, *"I wish I had fully engaged with the methods of data analysis (... which would have) allowed me to contribute more to the decision making"*.

Two out of the 12 poorest performing students had little programming experience and no subject-specific knowledge relevant to their project. Both worked on their team's poster and displayed the most dissatisfaction with their progress and experience. One Medicine student noted their ideas were "quickly dismissed", with their team members "believing that working with medical images would be too complex". Their project ultimately became considerably more complex, but leveraged only the interests of the two more experienced programmers.

An advanced project rooted in the discipline of the most experienced group members led to poor learning outcomes for the least experienced members. This was exacerbated by a (self-reported) non-compromising attitude of the expert members, leading us to believe that we must be more proactive in educating our students in group dynamics, especially in finding compromises.

Self-Assessment Mismatch

We also looked for clues in reflections that indicated that students overestimated their programming experience with ultimate negative consequences. Approximately one out of five students described themselves as having a fair amount of experience but reflected that they expected a basic computer science course or that they did not have sufficient time to learn enough to contribute to their project.

We continued this theme by analysing the reflections of the 12 students with a Connect 4 grade below 60%, which also revealed a group that is likely to need additional resources and support. Three of these students from Life Sciences and Medicine severely overestimated self-reported programming experience, which affected their group placement and resulted in a poorer experience and learning outcomes. This has led us to plan improvements to the self-reporting guidelines and the development of a scaffolded project with extra support.

Developing a Generic Framework for a Multidisciplinary and Multi-Skill Context

Based on our experience and analysis with multidisciplinary and multi-skill context, we created an adaptable framework to inspire fellow educators working in similar situations. Our framework is informed by the original module design elements and enriched by our data analysis, which suggested new interventions.

A multidisciplinary and multi-skill learning and teaching context represents an enormous opportunity. Students will be exposed to a wider range of problems, methods and ways of thinking. This context can become a vehicle for better outcomes. For our module, we used a flexible and varied design that was based on group project work. This design created an opportunity for bespoke learning and enabled us to bring research into the classroom.

Before undertaking thematic analysis, we have iteratively improved the model based on student feedback and, sometimes, anecdotal evidence. This chapter describes a deep dive evaluation that benefits any module from time to time. This systematic analysis of student reflections with other module data demonstrates that such an analysis can produce a wealth of new information and perspectives.

Using our case study, we draw generic lessons on key aspects that were uncovered by our analysis: supporting individual skills development, project group facilitation, and the overall narrative presented to students. The list indicates that our context requires a shift in focus and a different approach in data analysis than teaching a single-discipline audience with similar preexisting skills.

Individual Skills Development: Our Suggestions Based On Our Framework

The multi-skill context requires creating conditions for supported individual learning opportunities. Increased support requirement puts extra strain on resources but may be streamlined into a few well-defined learning tracks. For our modules, we use self-study of locally developed Python short courses that are normally taught to a postgraduate audience. Here are our suggestions for other teachers working with a multidisciplinary audience with varied prior skills.

- **Assume a base level of skill and be proactive in communicating this expectation clearly**. If possible, set a prerequisite for learning. For example, before project work starts, all students must have completed a specific introductory Python course.
- **Set individual learning with ample options for help**. Use inventive means to motivate learning, such as tournaments and coding challenges.
- **Use self-reporting to map skill level and monitor the outlier skill groups**, especially the least experienced students.
    - Not all students from the least experienced group will miss learning opportunities, but this group is most at risk. Pay attention to self-efficacy development opportunities, suitable project choice and division of work in the groups.
    - Include an opportunity for students to reflect with peers who closely match a learner profile to generate a self-efficacy-enhancing opportunities.
    - If meaningful participation of students in mixed groups cannot be guaranteed, consider establishing outlier groups, for example, a scaffolded project for coding beginners.

Project Groups Facilitation: Our Suggestions

This is a key area that was identified in our study. Project work facilitation is a large part of implementing group work and largely determines the quality of students' experience. It tends to be neglected compared with technical skills development. Our suggestions are:

- **Prepare students for self-assessment**. Ask for evidence for skills.
- **Assemble mixed groups** and aim for maximum distribution of disciplines and skills.
- Pay **extra attention to beginners' placement**. For example, a single beginner in an advanced group is not a good choice.
- Factor in **resources for group facilitation**. It can be equally demanding on staff time as supporting research aspects of project work.
- Set and **promote culture based on values facilitating effective group work**, and honest reflection with group peers.
- Plan for **systematic and transparent staff and student preparation for group work** and group work facilitation, including written materials; for example, group contract, guidelines for collaboration, project selection criteria, dividing work in meaningful and inclusive ways, reflection tools, appreciating, encouraging and including others, approaching difficult conversations, and a process for group check-ins.
- For open-ended projects, develop a **project choice rating framework** that helps to identify projects that provide appropriate motivation to all and do not lean heavily into the expertise of one or two individuals. Projects requiring advanced discipline-specific knowledge are not recommended. Projects should provide an opportunity to practice cross-cutting computing tools such as machine learning or high performance computing.
- Be extra careful to **manage engagement issues**. If a group member stops participating, reassure the remaining group members that a good outcome is attainable even with a smaller group, as the impact of low or variable participation can be demotivating. Fall back on clear guidelines for intervention.

Holding It All Together: Our Final Suggestions

The learning design in this context is not straightforward, and it is important that students understand why we build it this way. There are several themes running through the module that require students' attention. At the very least, students navigate the individual and group learning journey at the same time. Based on our experience and data analysis, the following points improve student experience and help them organise their thoughts.

- **Clear narrative and signposting of objectives during the module** – do not hesitate to share with your students why you organised the module in a particular way, including examples of what worked and what did not work in the past.
- **Share suggestions from previous cohorts** with your students and any other relevant testimonials.
- Students indicated that they enjoyed the difficult parts of the design the most. Apart from one task, these were not connected to any assessment. It leads us to speculate that it is a good design decision to **remove the burden of assessment and increase accessibility of the most challenging parts** of the design that are not connected with project work.
- Finally, **plan for module data analysis** and consider the type of questions that you need answered. We would not have been able to get the insights described in this chapter from the student feedback alone.

In Closing

In a few words, the multidisciplinary and multi-skilled context presents great opportunity for students to share experiences and learn together while working on a project. When projects are carefully scoped and group have access to dedicated advisors, the design of the project can be made appropriately ambitious while including all group members. It is essential that students' efforts must be matched by ample support and always accompanied by clear and reinforced guidelines for group work. Our experience suggests that when we pay equal attention to all three elements - facilitating individual development, supporting and guiding group work, and presenting a clear narrative - the multidisciplinary and multi skilled module can become a transformative experience.

References

Almond, Richard J. 2009. "Group Assessment: Comparing Group and Individual Undergraduate Module Marks." Journal Article. *Assessment & Evaluation in Higher Education* 34 (2): 141–48. https://doi.org/10.1080/02602930801956083.

Braun, Virginia, and Victoria Clarke. 2006. "Using Thematic Analysis in Psychology." Journal Article. *Qualitative Research in Psychology* 3 (2): 77–101. https://doi.org/10.1191/1478088706qp063oa.

Daniels, Harry, Michael Cole, and James V Wertsch. 2007. *The Cambridge Companion to Vygotsky*. Book. Cambridge University Press. https://doi.org/10.1017/CCOL0521831040.

ECRI, Early Career Researcher Institute. 2026. "Research Computing and Data Science Topics." Web Page. https://www.imperial.ac.uk/early-career-researcher-institute/learning-and-development/courses-by-programme/research-computing-and-data-science/.

Piaget, Jean. 1970. *Science of Education and the Psychology of the Child. Trans. D. Coltman*. Book. New York: Orion Press.

Thomas, JW. 2000. "A Review of Research on Project-Based Learning. San Rafael, CA: Autodesk." Generic.

Vygotsky, Lev Semenovich. 1978. *Mind in Society: Development of Higher Psychological Processes*. Book. Harvard university press.

Learning Together Across Modes: Online and On-Site Pair Programming in a Fusion Course

Beatrice Alex

Edinburgh Futures Institute, University of Edinburgh; School of Literatures, Languages and Cultures, University of Edinburgh

Clare Llewellyn-MacRae

School of Social and Political Sciences, University of Edinburgh

Pawel Orzechowski

Edinburgh Medical School, University of Edinburgh

Luke Stephens

School of Social and Political Sciences, University of Edinburgh

Introduction

"Sarah, can you scroll down a bit?" The voice appeared suddenly in Sophie's headphones, causing her to jump slightly in her chair. Just moments before, she'd been chatting face-to-face with Dr Llewellyn about Python syntax errors. Now her disembodied voice was helping her online partner navigate their shared code, while she remained physically present in the classroom just three feet away. Welcome to fusion teaching, where instructors become friendly ghosts and the boundaries between digital and physical learning dissolve in unexpectedly delightful ways.

In this chapter, we want to share our experience of developing and teaching an experimental and innovative pair programming course.[83] This course is non-traditional; we teach text mining to master's level students in fusion mode (occurring in a highly technologised classroom designed for teaching online and on-site students as a single cohort to involve all students as fully as possible)[84] in a two-day intensive session. We knew this would be challenging, but we also knew that this would be a great learning experience and a chance to do things differently. We present here an honest reflection of the trials, tribulations, and wins in what was ultimately a rewarding and satisfying experience. We didn't do everything perfectly, but we learned a lot, and we believe the students did too. Please read on and hopefully enjoy hearing about our experience.

Pair programming (Orzechowski, Blankinship, and Banas 2026; Desvages et al. 2026) is central to our course, but pairing students effectively in a fusion setting presents its challenges. It was important to us for our entire cohort to feel a sense of parity; this could not be an in-person course with people watching online. We wanted to develop fluidity between the modes. We worked hard at this and broke down the barrier between the room and online. We created a classroom in which both teachers and students can be vulnerable and open, so that we can grow from each other's mistakes, and communicate with respect.

When we first designed and ran *Text Mining for Social Research*[85], the Edinburgh Futures Institute's first fusion course at the University of Edinburgh, we knew we were stepping into uncharted territory (Alex et al. 2021). Fusion courses integrate both in-person and online students, requiring thoughtful interaction strategies to create a cohesive learning experience. While hybrid teaching models have been explored extensively (Beatty 2019; Raes et al. 2019) and so has the value of peer instruction (Porter et al. 2011), our fusion approach to intensive pair programming represents a novel application of these principles to collaborative coding education. With a team of four (three lecturers and one teaching assistant), we had the flexibility to experiment with innovative teaching strategies that bridged the gap between online and on-site students. This chapter presents our experiences and lessons learned when experimenting with three key interaction methods: *Negotiating Fusion-Pairs*, *the Ghost Helper Effect* and *Micro-interactions*.

[83] At the University of Edinburgh a "course" is usually a block of teaching (5-10 weeks) on a subject with assessment at the end, which awards academic credits. In your institution this might be called differently, e.g. class, subject, module or topic.

[84] See QAA Scotland - Examples of good practice

[85] This was a two-day intensive bootcamp course for 30 postgraduate students (15 on-site, 15 online). Each day consisted of four 1.5-hour-long sessions, with each session starting with a 15-minute introduction, an hour of coding in pairs and a 15-minute Q&A session.

Making Fusion Pairs Work

A reflection on what needs to be considered when pair programming in a fusion classroom

Pair programming is central to our course (Williams et al. 2000; Hannay et al. 2009; Hanks et al. 2011), but pairing students effectively in a fusion setting presented its challenges. We experimented with assigning students in same-mode pairs (on-site-on-site or online-online) and mixed-mode pairs (on-site-online), adjusting pairs every hour to help students feel part of a single, integrated cohort. Each new pairing required students to renegotiate key aspects of their collaboration: how they worked together, who took the lead, and how they interacted with instructors. This constant reshuffling reinforced the sense of a shared learning experience across both modes.

We encountered several challenges in this type of paired learning approach (audio issues, cognitive load, context switching). This could especially affect students when accessibility issues or other special circumstances are compounded or clash with the learning environment.

While students generally preferred mixed-mode pairs (see Figure 20.2) and reported feeling more connected to the broader cohort, this arrangement proved exhausting to both online and on-site participants. The background noise in the classroom, combined with the additional cognitive load students experienced as a result of this noise, can leave them feeling drained by the end of a session.



Figure 20.1: Different modes of paired students working together on programming exercises.



Figure 20.2: Mixed-mode pair programming in action with noise in background.

The size and acoustics of the classroom are key factors to consider when using pair programming in teaching. Background noise not only causes "Zoom fatigue" for online students (Bailenson 2021), especially in longer events like ours, but also affects the on-site students who wear headphones to connect to their mixed-mode buddy. Such overload issues disproportionately affect neurodiverse learners who may struggle with auditory processing or filtering competing stimuli. In our case, additional room facilities or even a quiet area outside of the classroom proved handy for such interactions. As a teaching team, we hadn't originally anticipated this need. We are known to improvise and have once used a "broom cupboard" as an overflow space to support mixed-mode pairs.

Unlike typical online interactions, where participants can selectively mute themselves, paired programming also demands constant verbal communication as students alternate between navigator and driver roles throughout exercises. This requirement for continuous dialogue means that small audio cues ("aha," "umm") become important elements that would make the experience worse if they were missing. The downside of not muting ourselves is that background noises cannot be avoided.

As teachers, we had to learn what the experience of an online participant is. When joining students online, we noticed how important the camera angles are. Online students like to see teachers close up but also what's happening in the classroom. When teaching hearing-impaired students online, they like to see the face of the lecturer close up for lip reading. When looking at code on a screen, the font size needs to be sufficiently big, and it is important to scroll in a controlled way, as otherwise it becomes difficult for the partner to follow. Often, the first few minutes for each pair's collaboration were taken up by making all those audio/visual adjustments, and then they were ready to go.

With several mixed-mode pairs in one room, there might also be a feedback loop (which sounds like a high-pitched squeak when a microphone picks up the sound from the speakers). This is something that needs to be managed carefully, and, again, additional physical space will help with that. In an hour-long session, such adjustments can take up another few minutes to sort out.

More broadly related to fusion teaching, addressing and talking to online students explicitly during the introduction of the course and throughout, and not just the ones, is vital to make them feel part of the group, as it is very easy to forget about them otherwise. Having personally participated in hybrid courses that prioritised those in the physical classroom, the online experience can feel isolating. Inertia and gravity pull teachers towards excluding online participants, if there are active efforts and structures to include them. For our text mining course, we rehearsed all possible scenarios of interacting with on-site and online students in the bigger group or in pairs (e.g. how do I interact with a student pair that is mixed-mode, on-site-only or online-only) and really thought carefully about what is important in each situation. We use an advocate for the online students in the room, a person with specific responsibility, who could be another teacher, a teaching assistant, or a student, to draw attention to any comments or raised hands from an online student that the teacher may have missed. This has the dual effect of including the online students in that moment but also reminding the teacher to check in the future. Making space to ask the online students for their comments is important; they become more active participants, and the students in the room feel like they are part of the cohort.

After several years of practice teaching using pair programming in this and similar courses, we are tending more towards online-only and on-site-only pairs, given that some of the challenges involved (e.g. classroom setup) are out of our control and are not something we can easily adjust during the course. In some ways, this feels like a failure; we really wanted to completely cross that boundary, but if this makes it harder for the students, we must adapt. We found that running the mixed-mode sessions only in the morning, when everyone is fresh, allows us to feel like one group and reduces the likelihood of Zoom fatigue later in the day.

Ghost Helper Effect and the Fluid Divide

A discussion on a fluid divide and not siloing students into two groups

In a traditional classroom, students can simply raise a hand and ask the teacher for help. However, in a fusion course environment, mixed-mode pairs introduced a new dynamic. We quickly learned that if online students needed help, support also had to come online. Otherwise, teachers ended up only communicating with people in the room. To address this, all teaching staff were online during pair programming sessions, creating what we called the "ghost helper effect". This allowed us to speak to students in both modes simultaneously, whether through direct conversation in the room or via webcams and headsets to ensure an equal experience.

This led to a surprising phenomenon for on-site students in mixed-mode pairs, suddenly hearing a voice in their headphones from one of the teaching team they had just interacted with in person a few minutes earlier, like a friendly ghost. At first, this unexpected presence was disorientating, diverting attention and requiring a cognitive shift (see Figure 20.3). To normalise the experience, we consciously used online helpers for all mixed-mode pair work. While initially some students were spooked by the fluidity of the "ghost" seeming to be everywhere at once, they quickly found this both friendly and helpful.

In taking this approach, it was important for students to feel a sense of parity in how the course was being delivered and to develop a fluidity between online and on-site. With staff being online and visible through their own cameras, they could speak to both groups directly through the same method. Rather than speaking to an online student through an on-site partner's webcam or relaying information to be passed on, this

ensured that all information was delivered simultaneously and created a sense of fluidity between students and staff.

This was strengthened by staff at times leaving the main classroom and speaking to students in different locations, for example, a smaller classroom. They effectively became 'online' as well, working to break down the divide between the two modes, while still communicating with students together. In blurring the boundaries, neither group felt staff were giving preference to one side or the other, and the "ghost helper" approach allowed us to support students seamlessly regardless of location.

For students, this fluidity created a sense of seamlessness between the two media, and while much thought went into facilitating this, for the students, it felt effortless. Some even shifted between modes during the course, starting the day online before coming on-site or the other way around. Allowing this flexibility showed that students felt at ease in either mode and that, while both had specific issues, neither was better than the other. For instance, at one point, electrical issues on-site meant that students in the room were unable to access their workbooks, while those online were unaffected. Therefore, while both modes could present challenges, in creating flexibility between them, students were able to adapt with little disruption.

Figure 20.3: The ghost helper effect.



Micro-Interactions

Micro-interactions are how we bring the learning into the wider classroom

When you teach a course in an intensive mode, you are all together for two very busy days. This helps with a feeling of connection and that we are all in this together. It exacerbates feelings of joy but also of frustration. Learning happens in the midst of these feelings, and we know we need to connect the course with those small moments of joy and frustration and allow them to be voiced. We wanted to encourage, celebrate, and capture these moments of learning and cohort camaraderie, and quickly fix mistakes we were making and habits that were annoying.

Below we will explain strategies we employed to make the classroom a place of sharing, exchange and growing together. Feedback was gathered not only for quality control, as with the typical "How did you like the course?" questions, but rather to continuously synchronise with the energy and mood of all the students on the course; we call this 'relentless feedback'. It has to be quick and fun and not distract the class from the core learning. We gamified these small moments of interaction, with methods for ensuring everyone has an equal and anonymous voice, avoiding some common techniques which can be quite exclusionary, such as when a teacher picks the student who raised their hand first, it is often the same student. We asked students to put crosses on a whiteboard of questions to show where their mood was, and we asked them to fire answers to verbal questions in the chat function of the online room; we call this a 'Chat Blast'. If a student picked up a mistake in the code, we asked them to circulate it to the room. If they had a good comment or insight during their learning, we asked them to bring it back up in the discussion sessions when we came back together as a whole group after pair programming. We adapted sessions when people were frustrated or tired, we voiced that these feelings

were valid and uncomfortable but can be part of learning, and we upped our energy when the energy in the room started to fade. We created a classroom in which both teachers and students can be vulnerable and open, so that we can grow from each other's mistakes, and communicate with respect (amplification of learning).

Relentless Feedback

We wanted to gather feedback as we went to have a quick and real-time reading on how the course is going and to be able to adjust our approach if needed. There is a trade-off between asking feedback on the spot or later: asking during the learning process can be quite disruptive, while if we ask later, students might not take the time to provide the feedback in the first place or might not remember how they felt earlier. This is especially important if we care about granularity and would like to know if, for example, a particular part of the course needs to be improved.

We envisaged a two-way feedback channel, where students say how they are feeling, but they also see how everyone else in the room is doing. We were also aware that anonymity will lead to more honest and useful answers. Another part was that we wanted it to be playful and not very constrained by the digital tool, where the little artefacts like handwriting, colour choice or exact position on the screen can convey personality and individuality.

We needed a quick and unobtrusive way to gather feedback, which meant that we could use it multiple times during the day. We also wanted to ask meaningful questions, which will help us guide the course delivery (not just "Do you like it") since we were aware of the complex interaction between whether a course is 'hard' and 'useful' (where 'hard' is fine, as long as it is 'useful').

We wondered whether we could create a feedback mechanism which is easier to complete than to dismiss/avoid, sort of like 'accept cookies' popups on websites make it easier to accept than to customise. So, we created a multiple-slider feedback mechanism, where students grade on a Likert scale multiple aspects of their learning experience, and all results are fully open but anonymous.



Figure 20.4: Feedback collected at the end of each badge.

For our fusion course, we could not find a tool which would deliver exactly what we needed at the time, so we used a shared digital whiteboard. The whiteboard would always start with several empty scales, and each student would be invited to type the letter X in the appropriate place on the whiteboard (see Figure 20.4). Since we gathered this feedback every 1-2 hours of the course (after each badge), we had very granular data about the two days of our course. In the animation below, you can see the way the mood in the room changed as the course progressed, which was very useful for us to see and respond to.

For completeness, this technique is not limited only to online or hybrid environments, as we also tried it on in-person bootcamp courses as a means of gathering and sharing feedback in the room (Figure 20.5).

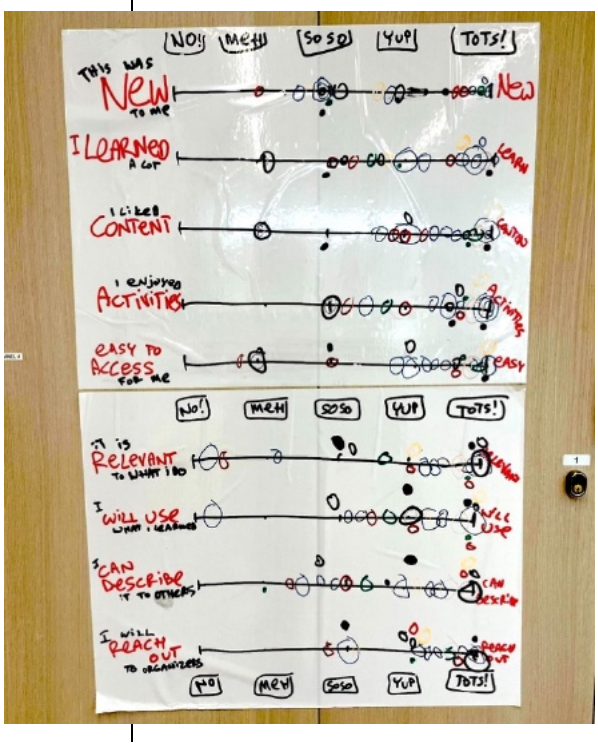


Figure 20.5: Feedback collected during a one-day intensive course on Hands-on Data Visualisation within Edinburgh Medical School at the University of Edinburgh.

Chat Blast and Other Micro-Contributions

Following the same philosophy as in the feedback above, we wanted to give students opportunities to check in with their own learning and the group and provide constructive feedback to the teaching team. Throughout the course, there were moments where we asked them to contribute questions that we would then discuss in the session after each pair programming exercise. This enabled the entire cohort to benefit from the discussion and answers provided. We also ensured that all student questions were repeated aloud during debrief sessions, while also encouraging students to share their own solutions via the chat.

At specific points in the course, we also asked all students to type a quick, micro contribution into the chat all at the same time, i.e. a Chat Blast (see Figure 20.6), to encourage engagement and shared participation. We took the Three Stars and A Wish approach, which is used by teachers in primary schools to help pupils focus on what is working and what needs improving (Larson, Trees, and Weaver 2008). In our case, we flipped this around and asked students to provide us with "Three stars and a wish" about the course in the chat of the video call. This enabled them to share their thoughts not only on what they liked about the course but also gave them permission to point out things that could be improved or things they struggled with.

Using the Chat Blast method, we exposed the fact that we were learning from the students as well. We were constantly adjusting the course in an agile way according to the feedback received from students when possible. We also exposed the fact that not everything goes right all the time, that we are human beings who can make mistakes and are learning how to adapt to each new group of students in a fusion teaching environment.

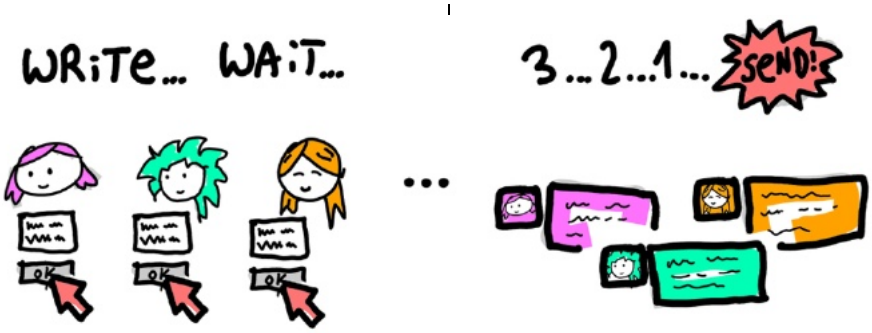


Figure 20.6: Chat blast with feedback using the "Three stars and a wish" method.

A Practical Guide for Fusion Educators

After two intensive days in the trenches of fusion teaching, we emerged with battle-tested insights that we wish we had known before we started. Here are our concrete recommendations for educators considering teaching pair programming in a fusion (or hybrid) setting:

Before You Begin:

- **Rehearse every scenario.** Practice switching between online and on-site interactions until it becomes second nature. We literally role-played "How do I help a mixed-mode pair when one student is confused?"
- **Audio is everything.** Invest in quality headsets and microphones for all teaching staff and test for feedback loops before the course begins. Poor audio will sink your fusion dreams faster than any pedagogical mistake.
- **Designate an online advocate.** Assign one person the specific job of watching for raised hands, chat messages and overlooked online participants. Without this, online students become invisible.

During the Course:

- **Start mixed, then adapt.** Begin with mixed-mode pairs when energy is highest, but don't be afraid to switch to same-mode pairs if fatigue sets in. Flexibility is not failure; it is responsive teaching.
- **Feedback must be quick and visual.** Our slider system took 30 seconds to complete and showed results immediately. Long surveys kill momentum.
- **Normalise the weird moments.** When your voice appears "ghost-like" in someone's headphones, acknowledge it with humour. Students need permission to find fusion teaching delightfully strange.

Technical Essentials:

- **Font size matters.** What looks readable to you will be too small for online partners watching code.
- **Avoid unnecessary scrolling.** Every rapid scroll or sudden page jump becomes disorienting for online partners trying to follow along. Move deliberately through content.
- **Camera angles are crucial.** Online students want to see both your face and the classroom; position cameras accordingly.
- **Have a backup space ready.** Whether it is a smaller room, or yes, even a "broom cupboard," you'll need overflow space for mixed-mode pairs.

The hard truth is that some days, same-mode pairs will work better than mixed-mode ones. Accept this not as defeat, but as responsive teaching. The goal is not perfect fusion; it is creating a cohort that learns together, regardless of location.

Our most important lesson? Fusion teaching requires the same vulnerability we ask of our students. When we admitted our mistakes in real-time and adjusted together, that's when the real learning, for all of us, began.

Conclusions

Fusion teaching challenges traditional classroom dynamics, but with thoughtful design, it can create an engaging and collaborative learning experience. By experimenting with different interaction strategies, we found ways to bridge the physical and digital divide between online and in-person students, creating an interactive and integrated cohort. Our lessons learned and practical insights are timely and valuable for educators designing hybrid courses, particularly in the post-pandemic education landscape.

Fusion teaching challenges traditional classroom dynamics, but with thoughtful design, it can create an engaging and collaborative learning experience. By experimenting with different interaction strategies, we found ways to bridge the physical and digital divide between online and in-person students, enabling effective pair programming partnerships across both modalities. The mixed-mode pairing approach, while challenging to implement in typical classroom environments due to background noise and cognitive load issues, demonstrated the potential for bridging physical and digital collaboration in programming education.

When logistical constraints made mixed-mode pairing difficult to sustain, our experience still provided valuable insights into preparing students for the increasingly common reality of distributed software development teams. Our lessons learned and practical insights are timely and valuable for educators designing hybrid programming courses, particularly in the post-pandemic education landscape, where graduates from various disciplines increasingly need coding and remote collaboration skills in their professional work.

References

Alex, B., C. Llewellyn, P. M. Orzechowski, and M. Boutchkova. 2021. “The Online Pivot: Lessons Learned from Teaching a Text and Data Mining Course in Lockdown, Enhancing Online Teaching with Pair Programming and Digital Badges.” In *NAACL-HLT 2021*, 138.

Bailenson, J. N. 2021. *Nonverbal Overload: A Theoretical Argument for the Causes of Zoom Fatigue*. *Technology, Mind, and Behavior*. Vol. 2. 1. American Psychological Association (APA). https://doi.org/10.1037/tmb0000030.

Beatty, B. 2019. *Hybrid-Flexible Course Design*. London, UK: EdTech Books.

Desvages, Charlotte, Brittany Blankinship, Umberto Noè, and Pawel Orzechowski. 2026. “Structured Group Work with Assigned Asymmetrical Roles and Switching: Lessons from Pair Programming Across Disciplines.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Hanks, Brian, Sue Fitzgerald, Renée McCauley, Laurie Murphy, and Carol Zander. 2011. “Pair Programming in Education: A Literature Review.” *Computer Science Education* 21 (2): 135–73. https://doi.org/10.1080/08993408.2011.579808.

Hannay, J., T. Dybå, E. Arisholm, and Dag I. K. Sjøberg. 2009. “The Effectiveness of Pair Programming: A Meta-Analysis.” *Information and Software Technology* 51 (7): 1110–22. https://doi.org/10.1016/j.infsof.2009.02.001.

Larson, K. A., F. P. Trees, and D. S. Weaver. 2008. “Continuous Feedback Pedagogical Patterns.” In *Proceedings of the 15th Conference on Pattern Languages of Programs*, 1–14. ACM. https://doi.org/10.1145/1753196.1753211.

Orzechowski, Pawel, Brittany Blankinship, and Kasia Banas. 2026. “Where Do i Even Start with Pair Programming in My Classroom? A Conversation with Seasoned Practitioners.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Porter, Leo, C. Lee, B. Simon, Q. Cutts, and Daniel Zingaro. 2011. “Experience Report: A Multi-Classroom Report on the Value of Peer Instruction.” In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education*, 138–42. https://doi.org/10.1145/1999747.1999788.

Raes, A., L. Detienne, I. Windey, and F. Depaepe. 2019. “A Systematic Literature Review on Synchronous Hybrid Learning: Gaps Identified.” *Learning Environments Research* 23 (3): 269–90. https://doi.org/10.1007/s10984-019-09303-z.

Williams, L., R. R. Kessler, W. Cunningham, and R. Jeffries. 2000. “Strengthening the Case for Pair Programming.” *IEEE Software* 17 (4): 19–25. https://doi.org/10.1109/52.854064.

A Practical Guide to Teaching Python as a Computational Tool in an Introductory Data Analysis Course

Serveh Sharifi Far

School of Mathematics and Maxwell Institute for Mathematical Sciences, University of Edinburgh
Serveh.Sharifi@ed.ac.uk

Ruini Qu

Edinburgh Business School, University of Edinburgh
R.Qu@ed.ac.uk

Stuart King

School of Mathematics and Maxwell Institute for Mathematical Sciences, University of Edinburgh
S.King@ed.ac.uk

Teaching Data Analysis and Programming: Does One Need to Come First?

Some introductory data science courses must teach both programming and statistics to students with no prior experience in either. Students may first take an introductory programming course; however, this is not always possible, and often programming must be integrated alongside data science concepts (Young et al. 2026). Teaching a data science or statistics course without a computational tool is not possible due to the amount of computations required in working with even small datasets. On the other hand, syntax-heavy programming courses may not immediately show relevance to data science and often struggle to engage students, who find it difficult to invest in abstract concepts without a clear sense of their practical application.

This chapter presents a structured, stepwise practical guide to introducing Python as a computational tool in an introductory data science course, allowing students to balance learning both technical skills (coding) with conceptual skills (statistical and data science thinking). The emphasis is on six fundamental elements to teach students to complement their data analysis learning in a process that supports a gradual transition from simple scripts to structured analysis. Each element introduces just enough Python to support statistical thinking, promote motivation, and empower inexperienced learners to use this tool for producing meaningful data analysis results.

Overview of Such a Course

This approach has been implemented by the authors in an introductory data science course for diverse learners at the MSc level in an eleven-week semester (King and Sharifi Far 2025). We suggest structuring such a course so that statistical concepts and methods are taught in lectures, while computer workshop sessions focus on applying the methods using Python. In the workshops, Jupyter notebooks that combine explanatory text and executable code can be used as a common and effective tool to practice coding (El Gemayel, Budiarto, and Bell 2026). This benefits beginners by keeping instructional material and code side by side and reducing the cognitive load that can come with more complex development environments. Furthermore, working on notebooks in the workshops can then be done in the pair-programming format to stimulate student discussion and peer learning (Orzechowski, Blankinship, and Banas 2026). To support learning, we also recommend integrating continuous assessment throughout the course. Using automated marking tools for coding tasks can provide timely, objective feedback and help students identify specific areas for improvement, which reinforces understanding and encourages regular engagement with the material.

Core Code Elements

Here we list the core code elements and Python functionalities fundamental to them, and the minimum set of inbuilt or module-specific functions needed to teach them. For each element, we describe its pedagogical goal and value, highlight common coding pitfalls observed in student learning, of which it can be helpful for instructors to be aware, and suggest real-world datasets to use that provide meaningful opportunities for practical application.

Element 1 - Foundational Coding: Expressions, Variables, and Basic Calculations

- **Goal**: Introducing students to how Python solves simple mathematical problems and handles expressions, assigns variables, outputs results, and applies simple built-in functions.
- **Pedagogical value**: This stage invites students to become comfortable with the Python syntax and environment. It allows them to store and reuse results to make their code more readable and flexible. Combining variables with simple functions introduces the core logic of programming. This stage builds confidence by showing how basic Python code can mimic a calculator while adding structure and repeatability.
- **Essential functions**: Creating lists of data using `[]`, assignment operator (`=`), arithmetic operators (`+`, `-`, `*`, `/`, `**`), printing results with `print()`, checking type of variables with `type()`. Some built-in mathematical functions, for example, `round()`, `sum()`, `len()`, `min()`, `max()`, `abs()`.
- **Data suggestion**: Using minimal lists and arrays of numbers and characters is helpful at this stage. Students can be invited to make a small list of their favorite fruits, colors, or a short series of numeric values (e.g., prices, scores).
- **Common pitfalls**: Misspelling variables or functions names; reusing variable names unintentionally; calling an unassigned variable (e.g., the notebook has defined some variables in a cell and students call these variables without executing the cell); forgetting to assign results of calculations to variables; misunderstanding variable type changes (e.g., overwriting a numeric/integer variable with a text/string).

Element 2 - Handling Data With `pandas`: Reading and Cleaning

- **Goal**: Equip students with the essential skills to load, explore, and clean datasets.
- **Pedagogical value**: It is essential for students to learn to work with structured data in the form of data frames. Introducing them to how to load datasets, explore their structure, and select or clean rows and columns is needed for visualisation, modeling, and interpretation.
- **Essential functions**: Importing `pandas`, functions such as `pd.DataFrame`, `pd.read_csv()`, `pd.read_excel()`, `head()`, `info()`, `describe()`, `loc[]`, `iloc[]`, selection of rows and columns with `[]` and operations on them (e.g., `df['col'].mean()`), handling missing data (`np.nan`, `dropna()`, `fillna()`, `replace()`), extra useful functions such as `groupby()`, `filter()`, `apply()`, `value_counts()`, `sort_values()`.
- **Real data suggestion**: Turtle Size dataset (Phillips et al. 2021) is a simply structured and helpful dataset which includes numerical and categorical variables. Penguins dataset (Gorman, Williams, and Fraser 2014) is another interesting and manageable dataset with some missing values for students to explore.
- **Common pitfalls**: Confusing `loc[]` with `iloc[]`; forgetting to save cleaned data in a variable; misunderstanding how `[]` behaves differently with rows and columns; forgetting that column names are strings when selecting them (e.g., `df["ColumnName"]`); forgetting to use a list when selecting multiple columns/rows (e.g., `dataframe.iloc[[5,1]]` vs `dataframe.iloc[5,1]`).

Element 3 - Visualising Data With `matplotlib` and `seaborn`: Explore, Compare, and Communicate

- **Goal**: Use visual tools to describe the data, explore distributions, trends, and relationships among variables.
- **Pedagogical value**: Visualisation helps students in conducting exploratory analysis of the data, identifying outliers, checking distributions, and spotting potential relationships between variables, and reinforces the connection between statistical concepts and their visual representations.
- **Essential functions**: Importing `matplotlib.pyplot`, then `plt.plot()` and `plt.show()` with `plt.xlabel()`, `plt.ylabel()`, `plt.title()`. Importing `seaborn` and using the general format of `sns.---plot(data=---, x=---, y=---, kind=---, ...)` for different `kind` of plots.
- **Real data suggestion**: House Sales in King County (Harlfoxem 2016) data is a manageable dataset for students to explore many ways of visualising house prices based on various numerical and categorical features. The Hotel Booking Demand dataset (Antonio, Almeida, and Nunes 2019) also provides a chance to visualise a hotel room prices and availabilities in Portugal based on several factors.
- **Common pitfalls**: Forgetting to label axes or add titles; confusing `kind` options in plots; overcomplicating plots by including many variables; difficulties with data restructuring if required for a plot; difficulty in understanding the connection between `seaborn` and `matplotlib` for beginners.

Element 4 - Summary Statistics: Describing Data

- **Goal**: Enable students to summarise variables and distributions and examine relationships between variables using descriptive statistics and correlation.
- **Pedagogical value**: Introducing summary statistics measures provides students with the language to describe data precisely. The concept of correlation offers an early opportunity to explore potential associations between variables, before modeling.
- **Essential functions**: Measurements of center and spread of data from `numpy` after importing it. The important functions are, `np.mean()`, `np.median()`, `np.mode()`, `np.std()`, `np.var()`, `np.quantile()`. Calculating Pearson linear correlation between two variables using `corr(method="pearson")`.
- **Real data suggestion**: Sleep in Mammals (Allison and Cicchetti 1976) and Animals Life Expectancy (Che-Castaldo et al. 2019) are interesting datasets about animals with various features in different types that can be used to encourage students to explore the pattern of how long these animals sleep and live.
- **Common pitfalls**: Applying numeric-only summary functions to non-numeric variables; ignoring or not checking for missing values; not inspecting the shape and possible skewness of the distributions.

Element 5 - Normal Linear Regression With `statsmodels`: Explain and Predict Relationships

- **Goal**: Provide students with tools to build linear regression models to explain relationships between variables and make predictions, using syntax that mirrors traditional statistical notation.
- **Pedagogical value**: Linear models are simple yet foundational, offering students a strong basis for deeper study of statistical modelling. Introducing the normal linear model, along with the binary linear model (logistic regression) if time permits, could be sufficient at this stage. Attention should be given to checking the model assumptions and its goodness-of-fit.
- **Essential functions**: Importing the library `statsmodels.formula.api`, using the `ols` function in the form of `model = smf.ols("y ~ x_1 + x_2", data=df).fit()` and the corresponding required functions, `model.summary()`, `model.predict()`, `model.resid`. Some useful plots are `sns.lmplot()`, `sns.residplot()`, and `qqplot()` for visualisation and checking the model assumptions.
- **Real data suggestion**: World Happiness Report ("The World Happiness Report," n.d.) data from most countries around the world and different years can be used to aim to model the countries level of happiness based on various measured social elements. Programme for International Student Assessment (Organisation for Economic Co-operation and Development, n.d.) data provides standardised exam scores for a large sample of students in different countries along with many social and educational indicators.
- **Common pitfalls**: Difficulty in coding categorical variables; in extending the model formula to variable names including spaces; and in inputting new data in the `predict` function or passing in wrong data types.

Element 6 - Machine Learning With `scikit-learn`: Classification

- **Goal**: Introduce students to basic supervised machine learning for classification problems, using intuitive models like k-Nearest Neighbors, decision trees, and ensemble methods, and reinforcing best practices in model evaluation.
- **Pedagogical value**: Introducing classification methods shows students how algorithms can learn patterns from labelled data to make predictions.
- **Essential functions**: From `scikit-learn` use `train_test_split` to prepare data for training and assessing models. Then apply different classification methods, using `KNeighborsClassifier`, `DecisionTreeClassifier`, `RandomForestClassifier`, `BaggingClassifier`, and `HistGradientBoostingClassifier`, which all need `fit()` and `predict()`. Evaluation of the classification can be done using `classification_report`, `confusion_matrix`, and a simple scatter plot like `sns.scatterplot()` is useful in visualisation.
- **Real data**: Pima Indian Diabetes dataset (Smith et al. 1988) includes various health indicators to use in modelling to predict whether participants would develop diabetes. Behavioral Risk Factor Surveillance System (Centers for Disease Control and Prevention, n.d.) data is available from several years and includes many health indicators and records of presence of many health conditions to use in different classification problems.
- **Common pitfalls**: Not applying the train/test split or mishandling its output; difficulty in choosing and applying the appropriate evaluation methods.

Depending on the course level and learning objectives, Element 6 may be included only if an introduction to machine learning falls within the scope of the course. Here, we presented classification as an example of a supervised learning method, although an unsupervised method such as clustering could also be introduced following a similar structure.

What Can Students Take Away?

By working through these core elements, students gain a practical foundation in both Python programming and data analysis. This progression helps students move from simple code scripts to structured analytical workflows, equipping them with the confidence and technical ability to engage with real-world datasets. By the end of this material, students should be able to implement core Python functions for data analysis independently and recognise how coding supports statistical thinking. The common pitfalls noted throughout are not mere technical errors, but can be important learning opportunities.

References

Allison, Truett, and Domenic V Cicchetti. 1976. “Sleep in Mammals: Ecological and Constitutional Correlates.” *Science* 194 (4266): 732–34.

Antonio, Nuno, Ana de Almeida, and Luis Nunes. 2019. “Hotel Booking Demand Datasets.” *Data in Brief* 22: 41–49. https://doi.org/10.1016/j.dib.2018.11.126.

Centers for Disease Control and Prevention. n.d. “Behavioral Risk Factor Surveillance System (BRFSS) Data.” https://www.cdc.gov/brfss/annual_data/annual_data.htm.

Che-Castaldo, Judy P, Amy Byrne, Kaitlyn Perišin, and Lisa J Faust. 2019. “Sex-Specific Median Life Expectancies from Ex Situ Populations for 330 Animal Species.” *Scientific Data* 6 (1): 190019. https://doi.org/10.1038/sdata.2019.19.

El Gemayel, Joseph, Arif Budiarto, and William Bell. 2026. “Notebook for Novices? Pros and Cons of Jupyter.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Gorman, Kristen B, Tony D Williams, and William R Fraser. 2014. “Ecological Sexual Dimorphism and Environmental Variability Within a Community of Antarctic Penguins (Genus Pygoscelis).” *PloS One* 9 (3): e90081. https://doi.org/10.1371/journal.pone.0090081.

Harlfoxem. 2016. “House Sales in King County, USA.” Kaggle dataset. https://www.kaggle.com/datasets/harlfoxem/housesalesprediction.

King, Stuart, and Serveh Sharifi Far. 2025. “Teaching Data Science to Diverse Learners: A Hybrid and Interdisciplinary Approach.” *Teaching Statistics*. https://doi.org/10.1111/test.70014.

Organisation for Economic Co-operation and Development. n.d. “PISA Dataset.” https://www.oecd.org/en/about/programmes/pisa/pisa-data.html.

Orzechowski, Pawel, Brittany Blankinship, and Kasia Banas. 2026. “Where Do i Even Start with Pair Programming in My Classroom? A Conversation with Seasoned Practitioners.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Phillips, Katrina F, Gustavo D Stahelin, Ryan M Chabot, and Katherine L Mansfield. 2021. “Long-Term Trends in Marine Turtle Size at Maturity at an Important Atlantic Rookery.” *Ecosphere* 12 (7): e03631. https://doi.org/10.1002/ECS2.3631.

Smith, Jack W, James E Everhart, William C Dickson, William C Knowler, and Robert Scott Johannes. 1988. “Using the ADAP Learning Algorithm to Forecast the Onset of Diabetes Mellitus.” In *Proceedings of the Annual Symposium on Computer Application in Medical Care*, 261.

“The World Happiness Report.” n.d. https://worldhappiness.report/.

Young, Robert S., Rebecca L. Colquhoun, Tiago A. Marques, Brittany Blankinship, Ozan Evkaya, and William P. Kay. 2026. “Sequential Vs. Simultaneous: Approaches to Learning Programming and Statistics.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Notebooks for Novices? Pros and Cons of Jupyter

Joseph El Gemayel

Computer & Information Sciences, University of Strathclyde
joseph.el-gemayel@strath.ac.uk

Arif Budiarto

Usher Institute, University of Edinburgh
arif.budiarto@ed.ac.uk

William Bell

Computer & Information Sciences, University of Strathclyde
william.bell@strath.ac.uk

Introduction

Computer programming has become a core skill for many disciplines, from biology to business, as well as computer science conversion degrees that are designed for students with no prior programming experience. In many contexts, programming is used to solve domain-specific problems. Students need to learn core programming concepts, while avoiding distractions that are associated with the nuances of particular Integrated Development Environments (IDEs) or software installation issues.

The demand for accessible programming education is particularly strong in programmes that attract students from diverse backgrounds who may not have studied computer science yet need to acquire coding skills to solve practical problems in their fields. Some of these programmes include aspects of Artificial Intelligence (AI) and Machine Learning (ML), which are rapidly evolving and highly valued by industry. These degrees attract students from diverse backgrounds, who may not have studied computer science, but are eager to explore AI & ML and their applicability in their work. Most of these programmes rely heavily on the Python programming language, valued for its simplicity and extensive ecosystem of AI/ML libraries.

Jupyter Notebook[86] has emerged as a popular tool for teaching Python for data science and AI/ML applications. A Jupyter Notebook integrates text comments, Python, and visualisation of results from running the Python segments. The visualisation includes values that are returned, as well as graphical displays of selected data. A user is able to modify a Python segment, re-run it and instantly evaluate the result, encouraging experimentation. However, beginners may be confused by hidden execution states and develop poor debugging habits.

This chapter explores Jupyter Notebook's strengths and weaknesses as a tool to teach computer programming to novice learners, highlighting how instructors and students can leverage its benefits while addressing its limitations. Although Python serves as the primary example in this chapter, Jupyter Notebook can be used with over forty programming languages.

What Makes Jupyter Notebook Appealing To Beginners?

The choice of a development environment can determine whether programming feels approachable or intimidating, especially for students encountering it for the first time. While professional IDEs are powerful, they may include features that distract a first-time programmer. In contrast, Jupyter Notebook provides a simple user-friendly interface, together with immediate feedback. This section explores some of the practical features of Jupyter Notebook that are appealing to beginners, including IDE functionality, collaboration and visualisation, supporting its use within an educational context.

Figure 22.1: Printing 'Hello, world' with a Jupyter Notebook.

```
[1]: # "Hello, world!" first application in Jupyter
     print("Hello, world!")

     Hello, world!
```

[86] https://jupyter.org/

An Easy-To-Use IDE

In professional Python programming environments, a developer normally creates a virtual environment and installs packages, which is driven by command-line tools. Jupyter Notebook provides a simpler interface, allowing Python to be written directly into a cell and executed with a keystroke. An example Python segment is given in Figure 22.1, where program execution is triggered by pressing *Shift+Enter* and the output is given below the `print` command.

As shown in Figure 22.1, there is no need to create a virtual environment, install packages or open a command-line interface during everyday use. Once Python, Jupyter Notebook, and relevant packages have been installed, the user is presented with a simple workflow. The instant evaluation of Python code makes programming feel less abstract and more welcoming to novices.

Working With Code in Pieces

One of Jupyter Notebook's defining features is its cell-based execution. Instead of writing and running an entire script at once, learners can break programs into smaller, manageable pieces. This modularity allows students to focus on one part of the code at a time and makes it easier to identify where errors occur.

The code in Figure 22.2 creates a list, defines a function to square all the numbers in that list, and then prints the results. Rather than placing everything in a single large cell, the program is split into three separate cells. If an error appears in one step, the student can rerun only that cell without restarting the whole Notebook.

The cell-based workflow encourages experimentation, where learners modify input values, rerun a function, and immediately see how the output changes. This reduces frustration, supports trial-and-error learning, problem-solving, and generally encourages novice programmers.

Collaboration Made Simple

Programming often requires teamwork. Jupyter Notebook supports collaboration seamlessly through platforms such as Google Colab. Students can share a link to a Jupyter Notebook, allowing others to edit or comment in real time, similar to working on a shared online document. For example, a group project for an AI/ML course can happen entirely within Colab, during which:

- one student writes the data pre-processing code,
- another tests different models,
- a third documents the process using Markdown cells.

Google Colaboratory (Colab)[87] supports the development of Jupyter Notebooks within a cloud platform, such that software does not need to be installed on a local PC. Collaboration becomes natural, and the focus remains on the problem being solved rather than on technical barriers.

```
[1]: # Cell 1: Create a list
     numbers = [1, 2, 3, 4, 5]

[2]: # Cell 2: Define a function
     def square_list(values):
         return [x**2 for x in values]

[3]: # Cell 3: Call the function
     print(square_list(numbers))

     [1, 4, 9, 16, 25]
```

Figure 22.2: An example of cell-based execution

```
[1]: import numpy as np

     # Create an array of numbers
     numbers = np.array([1, 2, 3, 4, 5])

     # Square each number
     squared = np.square(numbers)

     # Calculate the mean
     mean_value = np.mean(squared)

     print("Numbers:", numbers)
     print("Squared:", squared)
     print("Mean of squared numbers:", mean_value)

     Numbers: [1 2 3 4 5]
     Squared: [ 1  4  9 16 25]
     Mean of squared numbers: 11.0
```

Figure 22.3: Using the NumPy library.

[87] https://colab.research.google.com/

Using Libraries With Ease

One of the biggest challenges for beginner computer programmers is learning how to extend a language with external libraries. When using a Python installation on a local machine, packages can be installed using the `pip` or `conda` package manager. Python `pip` packages are installed into the system or often into a virtual environment to prevent conflicts from arising. Python can also be installed using Anaconda, which includes the Python interpreter, together with many data analysis `conda` packages, and Jupyter Notebook. Due to the more comprehensive nature of the standard Anaconda installation, it is unlikely that students will have to install additional libraries, except when working with more advanced or specialised packages. For example, the NumPy library is part of the standard Anaconda installation and is also present in the cloud installation of Jupyter Notebook, such as Google Colaboratory (Colab). This implies that a student can import NumPy, as shown in Figure 22.3.

Other common libraries used for AI/ML included in Anaconda are Pandas and Matplotlib, and further libraries are also available as `conda` packages that can be installed separately.

Common Pitfalls and Challenges

While Jupyter Notebook has become a popular entry point into programming, especially in data science and education, it includes a variety of hidden complexities that can hinder the learning process for beginners. Its interactive design and flexibility are strengths, but they also introduce subtle challenges that may not be immediately apparent to new users. This section outlines key pitfalls that often disrupt the early coding experience and offers insight into why they occur.

Hidden Kernel State and Memory Persistence

The Jupyter Notebook's kernel maintains an active memory state throughout a session, storing all variables until the kernel is restarted. This allows code cells to be executed in any order, where the order of execution defines the memory state rather than the order within the Notebook. Non-sequential execution implies that variables and functions may exist or vanish depending on the execution sequence. While the Jupyter Notebook persistent state can be convenient, it obscures the underlying logic of variable scope and memory management.

If the Jupyter Notebook kernel crashes or is manually restarted, all stored variables are lost. Beginners often misinterpret the resulting errors as coding mistakes, which actually stem from changes in the kernel's memory state. Students should be encouraged to run cells sequentially from top to bottom before saving or sharing a Jupyter Notebook to maintain a consistent state. Students should be reminded to restart the kernel and rerun all cells periodically to ensure that errors are not caused by leftover variables.

Relying on linear execution alone can discourage students from defining functions or classes, which would otherwise simplify the implementation and associated debugging. Students should be prompted to encapsulate repeated or complex code into functions or classes to improve readability, reusability, and debugging.

Markdown vs. Misunderstanding Code Cells

Jupyter Notebook's integration of Markdown and code cells enables rich documentation alongside executable code. However, this dual format can be confusing for beginners who are unfamiliar with Markdown syntax. When working with Notebooks that are created by instructors or collaborators, users may struggle to distinguish between instructional text and executable code, leading to misinterpretation or accidental modification of non-code cells. Students should be reminded that cell types can be changed at any time using the toolbar or keyboard shortcuts, and that Markdown cells are for text only.

Package Installation and Environment Conflicts

Installing external libraries is a common task in Jupyter Notebook, but the coexistence of multiple package managers, such as `pip` and `conda`, can confuse novice developers. Beginners may unknowingly install packages using one method while the Jupyter Notebook's kernel is tied to another, resulting in "missing module" errors despite successful installation. These issues are not coding errors but arise due to environment mismatches that can be deeply frustrating without a clear understanding of dependency management. Students should be advised to check the active kernel's environment and install packages consistently using the same package manager to avoid conflicts. It may be necessary to require that only one Python installation is present to prevent beginners from being confused.

Limited Debugging Capabilities

Unlike many full-featured IDEs, Jupyter Notebook provides limited tools for debugging programs. Error messages appear only in the cell where the code is executed, and there is no built-in support for features such as

breakpoints (pausing execution at a specific line) or stepping through code line by line to observe how variables change. As a result, beginners often try to find errors by adding `print()` statements or by repeatedly modifying and re-running cells. While this can work for small examples, it becomes difficult to manage when errors involve multiple cells or more complex program logic. Novice programmers should be encouraged to use small, isolated cells for testing code, where appropriate.

Misunderstanding of the `return` Command

Jupyter Notebook prints values that are returned from functions. This can cause students who are learning Python using Jupyter Notebook to falsely assume that there is no difference between `return` and `print`, such that they use `print` within a function to try to return a value and are confused when it returns None. Tutors should explicitly demonstrate the distinction between `return` and `print` early on and encourage students to test function outputs by assigning them to variables rather than printing values inside functions.

Best Practices for using Jupyter Notebook with beginners

Based on our experience using Jupyter Notebook in teaching novice programmers, we have identified a set of practices that help maximize its benefits while avoiding common pitfalls. The following recommendations are intended to guide both instructors and students toward using Jupyter Notebook more effectively as a teaching and learning tool.

Package Installation

It is recommended to use Jupyter Notebook through Google Colab, or to install it locally either within a full Anaconda installation or using the `pip` package manager (where a full Anaconda installation is typically preferred). To simplify possible package conflict issues, only one version of Python should be installed on a local machine which is associated with Jupyter Notebook. Students should be explicitly instructed to either rely on preinstalled `conda` packages that are associated with an Anaconda installation, or use `pip` packages if `pip` has been used to install Jupyter Notebook.

Modular Construction

Similar to regular Python programs that are constructed outside of Jupyter Notebook, Notebooks should be split into different files that are associated with different steps of a data analysis process. Each Jupyter Notebook should be short enough that it can be executed rapidly from the top of the file. Students can then avoid re-executing cells and being subsequently confused by the resulting output.

If a section of Python is needed often within a Jupyter Notebook, it can be defined as a function or placed in a separate Python file and imported as a module. Defining functions or classes helps students to focus on debugging the specific logic of their program rather than repeatedly rewriting common functionality.

Debugging

Students should be introduced to different debugging approaches with Jupyter Notebook, including the `print` command, `%whos`, `%debug`, `%pdb` and the front-end debugger.

While `print` allows specified variable values to be printed, all variables that are within the current scope can be printed by executing `%whos` within a cell. This provides a user with the summary of the current kernel state, in terms of the defined variables and their values.

If a cell fails to execute, the debugger can be started by typing `%debug` into the following cell which launches the `ipdb` command prompt. This allows a user to print the value of variables that are present in the kernel memory by typing `p` followed by the variable name. The available `ipdb` commands can be listed by typing `help`. The debugging session can be closed by typing `exit()`.

If a section of a Jupyter Notebook crashes and needs to be investigated, the debugger can be automatically launched on an execution failure by including:

```
%pdb on
```

before the cells under test, followed by

```
%pdb off
```

If the execution fails, the user will be provided with an `ipdb` command prompt automatically. The `pdb` commands are demonstrated in Figure 22.4.

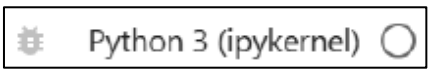


Figure 22.4: Using the pdb command within Jupyter Notebook.

More advanced debugging is accessible in Jupyter Notebook by using one of the kernels that supports the front-end debugger interface, which is documented in the JupyterLab Debugger online tutorial[88]. Similar to Visual Studio Code and other modern IDEs, breakpoints can be added by hovering over the line number within a code block. Once the program has paused, a user can inspect the values that are contained in variables and view the callstack.

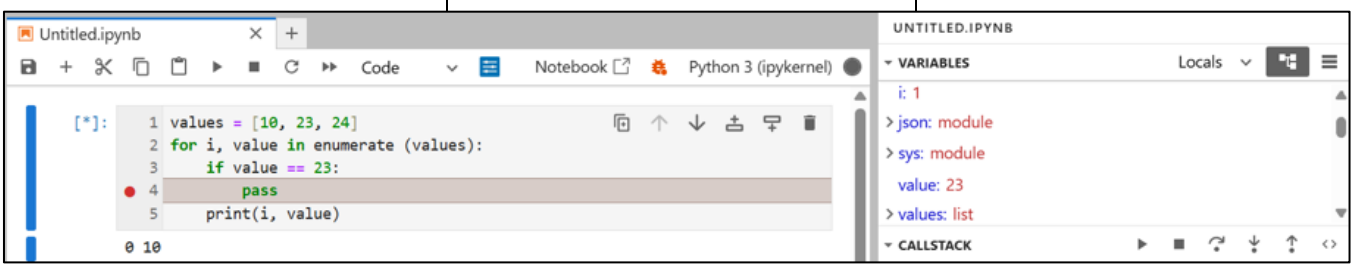
The front-end debugger is not enabled by default. When Jupyter Notebook is started, if the kernel supports debugging a grey bug icon is shown to the left of the kernel name as illustrated in Figure 22.5.

Figure 22.5: A kernel that supports debugging, where debugging is not active.



Debugging can be activated by clicking on the bug icon, which changes colour when activated. If the kernel is restarted, debugging becomes inactive again. When debugging is active, line numbers appear to the left of each line of Python code. Using the mouse to hover to the left of a line number causes a dark red circle to be displayed. Clicking on this red circle causes a breakpoint to be set for the specific line. When the code segment is run, the execution pauses at the breakpoint. The properties of values in memory can be viewed by enabling the Debugging Panel within the View menu. Paused code and associated Debugging Panel values are shown in Figure 22.6.

Figure 22.6: Paused execution and associated Debugging Panel values.



88 https://jupyterlab.readthedocs.io/en/stable/user/debugger.html

Markdown

Students should be encouraged to use Markdown once they have learnt enough Python to realise that Markdown is not associated with the Python programming language. To avoid confusion, students can use Python comments that are exchanged with Markdown formatted documentation later in a programming course.

Restarting the Kernel and Running Cells Sequentially

One of the most common difficulties beginners encounter in Jupyter Notebook is the confusion caused by non-linear execution. Since cells can be run in any order, variables may hold unexpected values, or code may appear to work only because of hidden state carried over from earlier runs. The Jupyter Notebook Kernel menu options can be used to address this. For example:

- "Restart Kernel and Clear Outputs of All Cells" clears the Jupyter Notebook's memory, ensuring no hidden variables remain, and removes all cell outputs for a fresh start,
- "Restart Kernel and Run All Cells" forces the Jupyter Notebook to execute in a clean, linear order, making sure all cells run correctly with no issues.

Encouraging students to restart and rerun the Jupyter Notebook periodically helps them avoid memory-related errors, reinforces the importance of execution order, and encourages good habits for reproducible coding.

Conclusion

Jupyter Notebook has remained a dominant tool for performing data analysis and AI/ML modelling. It allows the instantaneous execution of commands, such that software developers can see the effects of implemented Python code. It is accessible within the Google Colab cloud platform and may be installed locally within Anaconda or as a `pip` package. It provides students with an interface that is similar to a Databricks Notebook[89], which is used with large data samples.

While care is needed to avoid package manager conflicts and students should be guided on execution order and debugging practices, Jupyter Notebook's combination of code, visualisations, and text documentation offers a powerful platform for learning and experimentation. By leveraging these features thoughtfully, educators can make programming more accessible and engaging, opening the door for students from diverse fields to develop computational skills and confidence.

[89] https://docs.databricks.com/aws/en/notebooks/

Bridging Languages: Teaching C to Python Novices

Sam Skipsey

School of Physics and Astronomy, University of Glasgow
Samuel.Skipsey@glasgow.ac.uk

Gordon Stewart

School of Physics and Astronomy, University of Glasgow
Gordon.Stewart@glasgow.ac.uk

Jeremy Singer

School of Computing Science, University of Glasgow
Jeremy.Singer@glasgow.ac.uk

David Cutting

School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfast
d.cutting@qub.ac.uk

Introduction

This chapter explores challenges associated with learning a second programming language, in particular for a population of students who are not specialising in computing. In the context of our case study, the second language challenge is compounded in two ways: firstly, the programming paradigm shifts from a high-level managed interpreted language to a low-level unmanaged compiled language (i.e. from Python[90] to C[91]), and secondly, the development environment shifts from a browser-based notebook to a terminal-based text editor (i.e. from Jupyter[92] to an editor such as Vim). We highlight common misconceptions experienced by students during this transition process and discuss potential fixes, in particular the use of a *bridge language* to ease the transition from high-level to low-level paradigms.

About Us

Two of the authors of this chapter — Sam Skipsey and Gordon Stewart — comprise two thirds of the staff teaching the P2T course that forms much of this chapter's basis. Sam has a background in Theoretical Physics, although they have always been interested in computing as a topic; they might well have ended up doing a computer science undergraduate course had things gone differently. Gordon is a computer scientist and software engineer by training, who now works as a research IT specialist. The other two authors are based in computer science departments and are involved with teaching programming to computer science students directly. They also have interests in computational thinking.

Physics at The University of Glasgow

The rise of Python as the most popular programming language in the world has been coupled with the emergence of Project Jupyter and its notebooks as an extremely popular programming interface in many disciplines. It is now frequently the case (writing in 2025/26) that the only experience of coding that novice programmers may have involves "Python in Jupyter", and so naturally this becomes their sole model for how programming works. The ubiquity of Jupyter also means that some students may only know "Jupyter" as a synonym for another hosted development environment, such as GitHub Codespaces[93].

As a consequence of arguments similar to those described in (Balreira, Silveira, and Wickboldt 2022) and elsewhere, Python has displaced other languages in many university courses, not least in physics curricula. Broadly, the suggestion is that Python's perceived relative "simplicity" in syntax allows students to focus more directly on the actual process of "programming", rather than getting hung up on syntactical complexity as in (say) C. In the University of Glasgow's School of Physics and Astronomy, students encounter programming in their first-year labs when they use short Python snippets in Jupyter notebooks to process and analyse lab results; students subsequently receive a more formal introduction to Python in Jupyter during their second-year labs. In the rest of this paper, we will use "Jupyter" as a shorthand for "Jupyter with a Python kernel"; some of the aspects here are related to Jupyter itself, but others are dependent on the particular kernel (and thus the language) used, so we wish to be clear about the conflation here.

These two lab-based experiences are, for many Physics students, their very first introduction to programming and, indeed, their first introduction to the concept that programming is important to the practice of physics as a discipline. We've noticed that this sometimes requires a significant adjustment to the students' mental image of what it means to be a physicist in the modern world, which seems to be set by prior experience in

[90] https://www.python.org

[91] https://9p.io/cm/cs/who/dmr/chist.html

[92] https://jupyter.org

[93] https://github.com/features/codespaces

primary and secondary education as "someone who does experiments and knows advanced mathematics". Students are also expected to take mathematics courses in their first two years at university, but because they expect this as part of their image of the subject, they do not dispute the necessity of this; they may, however, dislike other aspects, for example the extreme rigour of some proof requirements in mathematical processes. By contrast, a significant fraction of physics undergraduates in their first year do find the introduction of computing topics both surprising and, if not unwelcome, then at least not well justified.

"P2T" – A Specialised Course in C for Physicists

In the second semester of year two, Physics offers the course "P2T[94]: C Programming Under Linux", which introduces lower-level programming in the C language[95], the use of the GNU/Linux operating system, and command-line tooling for debugging, build orchestration and version control. This course is a prerequisite for students on the Theoretical Physics track, who have the programming background mentioned in the previous section; meanwhile, approximately half of the cohort each year is made up of second-year students from the School of Computing Science, who begin with a much stronger background in programming concepts, including experience of assembly language. In fact, P2T is a small (10 credit) course designed to stand alone with no formal prerequisites and thus is available to any student; in practice, however, only a few students outside of physics and computing science choose to take it.

When first developed, at the turn of the millennium, P2T was considered a rather specialised course and was delivered to a small cohort. Today, mostly due to interest from computing students, it has grown to around seventy students a year; for comparison, the total number of students studying second-year physics and computing science is approximately 180 and 330 respectively.

We have observed that the difficulties experienced by the physics cohort in P2T have changed as their prior experience has become dominated by Jupyter. In this chapter, we will discuss how this relates to the mental models formed when learning to program in Jupyter, and their differences from the models needed for lower-level programming. We will also suggest some potential bridge languages that would ease the transition and help students to develop better mental models.

Misconceptions + Difficulties

This section comprises a brief list of the most common conceptual challenges we observe in students used to Jupyter when learning C in P2T. Quoted text is paraphrase from a genuine student question, other titled sections are summaries of the general area of misapprehension. There is no implication in the ordering of the examples.

1. "If you have a test (e.g. 3 < 1) which is false, does that make the compiler stop or the program stop?"

   This is quite a subtle distinction, which is less evident perhaps when learning first in an entirely interpreted interactive programming environment such as Jupyter, where code-blocks themselves obscure the flow of execution, let alone the distinction between compilation / interpretation and execution itself.

2. Source Code == Executable Code

   More generally, we often observe confusion between a program's code, its executable, and a process which represents a running instance of the program. The very first C exercise in the P2T labs walks students through the process of fixing some provided code, introducing an iterative sequence of compile-fix-repeat, and finally executing the compiled program. Despite this, towards the middle of the course we usually find that a significant number of students start trying to execute their source code, often after compiling it. In the case of P2T, this confusion is exacerbated by the fact that students are introduced to shell scripting in Bash, an interpreted language, around the time they develop this confusion.

3. Import == #include

   In order to do most useful things in C, we start students off with the magic incantation `#include <stdio.h>` and explain that this enables them to use functions that interact with the terminal, such as printing out text or reading input. We do allude to the fact that `#include` only provides half of the functionality that `import` does in Python, but when we introduce the full concepts of header files and libraries later in the course, students find this hard to reconcile with their idea of `import`. In particular, the need for a linking step is an alien concept.

4. Types

   Types represent a significant learning curve for our students, especially those who are entirely new to programming, or who been introduced to Python via the physics educational flow (despite being encouraged to use NumPy almost immediately in that case). Students' misapprehensions in this category are varied, ranging from the difference between variable declaration and assignment (in C and more strongly-typed languages, declaration requires the type of a variable to be specified, while assignment does not) through

[94] University of Glasgow short-codes for courses are almost opaque for historical reasons, but this was originally short for "**P**hysics level **2**: **T**heory".

[95] This is true of the 2024/25 iteration of the course, which was in place when we wrote the first draft of this chapter. We'll discuss which choice we made for 2025/26 at the end of this chapter.

allowable type conversions and the specific differences between different kinds of numeric scalar types, to uncertainty about the role and use of derived types.

5. CStrings and CArrays == Lists… and C types "are objects"

   In a sense, this is a subcategory of misconception 4, but it feels distinct because Python so strongly centres dynamic lists as the default container with a syntax strongly resembling that of C's decidedly static and strongly-typed arrays. This results in a number of conceptual problems for students either directly — trying to append to, slice or copy-by-assignment arrays — or indirectly trying to apply Python's "for-each" loop semantics to C via implicit array iteration (that is, writing a loop as a "for each i in CArray").

6. Scope and Lifetime / Encapsulation

   Python does have rules for both the scope of name bindings and the lifetime of allocated variables, albeit via garbage collection in the latter case, but the pragmatic way in which Python is taught in first and second year — and moreover the confusing way in which Jupyter cells interact with these concepts — means that students have not been exposed to these prior to starting P2T.

Mental Models

None of the above misconceptions are the students' fault: most are perfectly reasonable generalisations from an initial programming experience involving only Python in Jupyter, which naturally insulates developers from some of the underlying mechanics required to translate a series of high-level source code excerpts into a program that a computer can actually execute (Johnson et al. 2022).

Blaming the Snake or the Planet

It's important to spend some time teasing apart the relative contributions of the notebook context and the language itself in forming these misconceptions. As one of the authors has noted previously (Singer 2020), there are specific pedagogic *disadvantages* to the notebook approach which have been understood for some time, and long pre-date the advent of Jupyter; one of the authors remembers some of these issues arising when using *Maple* in the late 1990s!

Broadly, we observe that notebooks in particular make it very hard for students to develop mental models of the flow of execution through code, and the way in which state is transformed in this process. There are notebooks — for example, the *Pluto* notebook implemented in Julia — which partially fix this problem. Pluto addresses this issue by specifically re-executing *all* cells by default, in order, when one cell is modified. However, Jupyter remains the mainstream choice of notebook platform, by sheer weight of deployments. Thus, it may be obvious that it is the use of Jupyter, not Python, that is the primary cause of misconception 1, and that Jupyter contributes to some significant extent to misconception 6.

Misconception 2 may also be caused by the use of Jupyter to a greater extent than the use of Python. Students also find it hard to reason about files as objects, a well-documented issue as increasingly locked-down operating systems prefer to present search tools and containerised applications with their own local files, rather than the OS's own filesystem (Chin, n.d.)[96]. The use of Jupyter correlates with this, as notebooks are both seen as more natural for students used to browser interfaces (as has become increasingly common over the 2010s and 2020s) *and* themselves promote that same disconnection of content from location. In this sense, we believe that misconception 2 is promoted less by the fact that interpreted languages are not translated into a permanent, separate, executable artifact, and more because even the concept of code as living in static, separate files is elided by the interface. Of course, this issue might also be raised with any REPL, but the image of the modern web notebook as "just another browser app" surely does not help matters.

Misconceptions 3, 4 and 5, conversely, clearly originate with the language itself, or at least with the pedagogic approach adopted by courses the students have previously undertaken.

A "Whorfian" Third Way

Pedagogically, the programming languages we teach students in a specific discipline should be chosen to promote particular modes of thought. Pragmatically, we are also pressured to teach languages which provide students with employable skills, and these two goals may not perfectly overlap. Esoteric programming languages, for example, can help to develop specific problem solving or mental models in programmers (Singer and Draper 2025), but are by definition not widely used.

For our physics and computing science cohorts, there are therefore different reasons to wish to teach C in the first place:

Physics students:

- To develop mental models of low-level interfaces, important for those students who will develop experimental skills with electronic data taking, or hardware design; for example, the design of particle detectors may involve firmware development and FPGA programming.
- To give experience in performance programming in a language in which many core scientific libraries are still written.

Computing students:

- To develop mental models extending their knowledge of machine behaviour from assembly courses

[96] Tangentially, we also see students confused by the "content" of a file not being a function of its filename suffix: an internalised Windowsism.

studied in first year, and feeding into low-level topics covered at honours level, such as operating system and compiler design.

- To appreciate that the Python runtime abstracts away many aspects of program behaviour, including dynamic memory management.

As such, appropriate mechanisms for developing those mental models may differ between the two cohorts. We explore two potential 'bridge' languages to scaffold the mental models for students before their introduction to C: one already existing for computing science students, and one proposed for physicists that may even be able to supplant C for many of these students.

Bridge Language: Java

In the School of Computing Science at the University of Glasgow, students learn Python in their first year of undergraduate study, alongside a specialised assembly language designed specifically for teaching. In their second year, students undertake several courses using the Java language, most notably a course on Object-Oriented Software Engineering. By the time they begin P2T in the second half of this year, they have been exposed to multiple languages for several months, and have started to develop the agility needed to adapt their mental models appropriately for whichever language they happen to be using at the time (Tshukudu, Cutts, and Foster 2021).

The choice of Java may be particularly useful as a bridge between the conceptual frameworks implied by Python and those required for efficient development of code in C and C-like languages. In some ways, we can consider Java to be a halfway house: like Python, it is deeply object-oriented (in the C++ sense), to the extent that there are classes that wrap even the primitive fundamental types; like C, it is strongly-typed and its default containers are statically-sized, although Java does provide dynamic, heap-allocated containers in its standard library. Furthermore, Java syntax superficially resembles that of C and C++, further easing the later transition to C.

Alternative Bridge Language: Julia

We suggest that, for physicists in particular, there are other languages which are equally or indeed more compelling as bridge languages for low-level concepts.

Julia (Bezanson et al. 2014) is a just-ahead-of-time (JAOT) compiled, strongly-typed language significantly influenced by R, MATLAB and other languages oriented towards engineering and statistics. This influence means that much of the notation is designed to be familiar to mathematicians, and functionality for array programming concepts and statistical data processing is built in to the language, rather than being a feature of a supplementary package like NumPy. This has led to Julia becoming the teaching language of choice for a number of quantitative disciplines — for example, the University of Michigan features Julia in its theoretical courses in robotics (Grizzle, n.d., and others) — and has also seen support for it grow in other disciplines, including high-energy physics (Stewart, Graeme Andrew et al. 2025).

Indeed, one of the issues with the arguments for Python and against performance languages, such as those given in (Balreira, Silveira, and Wickboldt 2022), is that they beg the question, assuming that by definition a "performance" language must also be inherently less syntactically and semantically natural than an "intuitive" one (such as, it is argued, Python). This is neither obvious, nor actually true in the general case; C and C++ are arguably very unapproachable languages, but this is a consequence of their age and accretion of features more than it is due to their performance characteristics.

The power of Julia as a bridge language from Python is that one can gradually add more layers of subtlety and complexity as students develop their mental models. The equivalent process in Python requires learning large

external modules such as NumPy, which effectively provide an additional domain-specific language wrapped in Python for dealing with these concepts. In Julia, such concepts are supported within the core of the language itself.

A common pattern in development in Julia is to write functions in a general sense first, with no type specifiers on their parameters:

```
"quotient(numerator,
denominator) : Quotient of two
values"
function quotient(numerator,
denominator)
    numerator / denominator
end
```

Then, when necessary, one can specialise methods of the function for particular cases:

```
"quotient(numerator::Integer,
denominator::Integer) : Quotient
of two Integers - returns a
Rational"
function
quotient(numerator::Integer,
denominator::Integer)
    numerator//denominator
end
```

(Here we specialise `quotient` for any concrete `Integer` type variables to return a `Rational` fraction, rather than a simple division.)

This provides a natural way to introduce types and their relevance to students. In fact, internally Julia always specialises a function invocation by the types of the argument, and we can also introspect this if we want to demonstrate this fact to students.

Of course, Python does provide *type hints* as an optional feature from version 3.9. Unlike true type constraints, however, Python's hints have no effect on the Python interpreter itself; they are annotations to be parsed by external linters, or to act as additional documentation for the human reader. The actual output of the Python interpreter itself is unchanged by their introduction.

Furthermore, as almost all of Julia's standard library is itself implemented in Julia, Julia's introspection tools can provide a way for learners to look under the bonnet and explore the internals of the language. Unlike Python, there are almost no black-boxes that we can't examine. For example: the `@less` macro allows the student to look at the source code for any function that is loaded from a file in the current session. While this doesn't work for things defined in the REPL itself, there are packages that provide this functionality should it be needed.

```
@less isodd(3)

> isodd(n::Real) = isinteger(n)
&& !iszero(rem(Integer(n), 2))
```

(In fact, a view of the file in which this is defined, with the above line highlighted.)

Teaching performance programming in Python is — by contrast with teaching fundamental programming concepts — quixotically harder, in that much of the answer is to use Python fundamental constructs as rarely as possible. Whilst algorithmic design plays some part, the fact that we can make an implementation orders of magnitude faster by using NumPy intrinsics instead of for-loops muddies the lesson that we need students to extract from the experience. By contrast, in both C and Julia, implementations in the base language are about as fast as they can be. Efficiency is almost always gained by either algorithmic improvement or, in extremis, by making use of deep systems knowledge (cache-width, layout of data in memory, etc.) which is mostly language-agnostic, and can be relegated to a later course.

Because Julia *is* JAOT-compiled, however, lower-level representations are always available to the programmer, which can be useful for pedagogic purposes; something which is not possible with Python. That is, we can always use the `@code_` family of macros to expose what Julia code is compiled to, in an active session, at various levels in the compilation process, including at that of the native assembly code (via `@code_native`). For example, this lets us show the significant changes in code generation when switching between floating-point and integer types for mathematical operations, or how tail-call recursion is optimised to more efficient loops!

As a result, Julia is both a more complete bridge-language towards low-level programming concepts than Java (or directly C) *and* a more suitable *single* language for teaching students in physics and other mathematics-oriented disciplines than Python + NumPy (+ Numba + ...).

Of course, Julia is not a perfect language. In terms of popularity, it is still rather specialist; it is considerably less popular than Python, and less popular than Python + NumPy in physics circles. This is partly attributable to its relative youth — only 14 years to Python's 35 — but also due to network effects in language adoption. That youth, and a small development team relative to Python, also means that Julia's development cycles can still introduce more significant changes than the relatively stable Python 3. (Python, of course, passed through this phase during its 2 to 3 transition, when it was a similar age to Julia now.) This presents some difficulty in maintaining static teaching materials for some topics; automatic differentiation tools, for example, have been in some flux for the past year or so, due to changes in the language internals.

Alternative Bridge: Just Drop Jupyter

As roughly half of the misconceptions we note in this article are at least somewhat due to the abstraction and disconnect that Jupyter itself brings to the coding experience, a third and perhaps less dramatic choice might be simply to teach Python as a native language in a terminal context.

The Python interpreter is ubiquitous, especially now that Microsoft's significant interest in the language has led to it being available to script Excel workbooks; in MacOS and Linux contexts, of course, it is almost always available as a system component as well.

Writing Python code in actual files enforces structural discipline that isn't present when using notebooks, and also provides at least some reinforcement of the difference between source code and the thing that's actually doing the work. Indeed, one can wave at `.pyc` files if we wish to show a transformed output.

Other misconceptions in our list are not ameliorated by this approach, of course; in particular, misconception 4 is only partially addressable, and 5 is arguably intractable as C and Python simply have different object models. However, in a physics context, Python is almost always used with an implicitly-loaded NumPy package to handle scientific data. NumPy, in contrast to Python, cares deeply about types, and also enables new iterative constructs not found in the base language, which are broadly those constructs typical of array programming languages.

Conclusions

An important step towards students reaching competency in a programming language is to develop a mental model of the way in which the code one writes becomes a series of instructions that a computer can execute. While environments such as Jupyter undoubtedly provide an accessible interface to programming tools, this accessibility can come at the cost of rendering the internal workings of the languages opaque, which can make it harder for students to gain the insight necessary to progress to more advanced domains. Rather than attempt to move straight from interactive Python development in Jupyter to a low-level language such as C, it may be fruitful first to introduce a bridge language that allows students to explore more advanced aspects of programming within a modern environment.

The choice of bridge language can be discipline-specific, however, since different disciplines program in different contexts. Whilst there are arguments for languages such as Java for computing science students, we suggest that choosing superficially more "esoteric" languages, such as Julia, may be a better pedagogic fit for physics students. If that suggestion is considered too radical, the alternative of simply teaching Python *without* Jupyter may help to ground student understanding in ways useful to their future development. This is particularly true in a scientific context where "Python" is actually taken to mean "Python with some combination of NumPy / SciPy / Pandas / <insert your scientific library of choice here>".

Postscriptum

After this chapter was drafted, the decision was made to change the contents of P2T. The current version, which is in progress at the time of writing this postscript, broadly follows the "Just Drop Jupyter" approach. Whilst the pedagogic advantages of Julia were recognised by the relevant committees, it was felt that retaining a through-line with Python was more generally useful, and also allowed a wider subsection of academic staff to be able to teach the course.

In exchange, the course now covers software engineering concepts more substantially, with lab material on unit testing, performance profiling and reproducible code and packaging now provided. The performance profiling aspects have been the most challenging, in overcoming the intrinsic "just translate it to NumPy" problem, but we have taken the opportunity to introduce the physicists to some light conceptual exercises on the scaling of different data structures as part of this.

Interestingly, immediate feedback from the physics students who had taken the previous version of the course has been mixed: those who felt the need to comment mostly argued that learning a performance language like C had been valuable to their deeper understanding of programming as a discipline. It is, of course, too early to say if this is a wholly successful change, although interim feedback has been generally positive… except, again, for a small number who were hoping to learn C!

References

Balreira, Dennis G., Thiago L. T. da Silveira, and Juliano A. Wickboldt. 2022. “Investigating the Impact of Adopting Python and C Languages for Introductory Engineering Programming Courses.” *Computer Applications in Engineering Education* 31 (1): 47–62. https://doi.org/10.1002/cae.22570.

Bezanson, Jeff, Alan Edelman, Stefan Karpinski, and Shah. Viral B. 2014. “Julia: A Fresh Approach to Numerical Computing.” *SIAM Review* 59: 65--98. https://doi.org/10.1137/141000671.

Chin, Monica. n.d. “File Not Found: A Generation That Grew up with Google Is Forcing Professors to Rethink Their Lesson Plans (Internet Archive Link).” The Verge. https://web.archive.org/web/20210922124725/https://www.theverge.com/22684730/students-file-folder-directory-structure-education-gen-z.

Grizzle, Jessy. n.d. “Notes for Computational Linear Algebra.” GitHub. https://github.com/michiganrobotics/ROB-101-Textbook-Computational-Linear-Algebra/tree/main.

Johnson, Fionnuala, Stephen McQuistin, John O’Donnell, and Quintin Cutts. 2022. “Experience Report: Identifying Unexpected Programming Misconceptions with a Computer Systems Approach.” In *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1*, 325–30. https://doi.org/10.1145/3502718.3524775.

Singer, Jeremy. 2020. “Notes on Notebooks: Is Jupyter the Bringer of Jollity?” In *Proceedings of the 2020 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, 180–86. ACM. https://doi.org/10.1145/3426428.3426924.

Singer, Jeremy, and Steve Draper. 2025. “Let’s Take Esoteric Programming Languages Seriously.” In *Proceedings of the 2025 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, 213–26. Onward! ’25. ACM. https://doi.org/10.1145/3759429.3762632.

Stewart, Graeme Andrew, Moreno Briceño, Alexander, Gras, Philippe, Hegner, Benedikt, Hernandez Acosta, Uwe, Gal, Tamas, Ling, Jerry, et al. 2025. “Julia in HEP.” In *EPJ Web of Conferences*, 337:01266. EDP Sciences. https://doi.org/10.1051/epjconf/202533701266.

Tshukudu, Ethel, Quintin Cutts, and Mary Ellen Foster. 2021. “Evaluating a Pedagogy for Improving Conceptual Transfer and Understanding in a Second Programming Language Learning Context.” In *Proceedings of the 21st Koli Calling International Conference on Computing Education Research*, 1–10. ACM. https://doi.org/10.1145/3488042.3488050.

Dungeon Crawlers and Anarchists: Co-Designing Programming Teaching Experiences

Francisco Queiroz

School of Design, University of Leeds
f.queiroz@leeds.ac.uk

Maeve Murphy Quinlan

Research Computing, University of Leeds

Introduction

You find yourself in a large room with no windows; the air is still, and nothing suggests the passing of time. The door closes behind you, and it won't open for another hour or two. As you stand in front of the room, you realise there is nothing between you and what seems to be a party of the most intimidating, unpredictable creatures.

Students.

Fear not, though. You are about to learn how to deal with this kind of situation. Dealing with students' expectations (and your own) can be hard, but we believe that by working together, you will be able to deliver the learning experience that gets everyone to the next level.

This chapter is inspired in equal parts by tabletop role-playing and anarchist pedagogies. It was created to help you in structuring the learning experience for your course. More specifically, we present steps you can take during an initial or preliminary learning activity for a course, preparing material that will help you:

- get to know your students, their expectations and shortcomings
- organise teams for collaborative group work
- relate learning sessions and activities to the course's learning outcomes
- structure the sequence of learning activities
- share resources

This system is inspired by variations of Role-Playing games that do not require a Dungeon Master as mediator, such as various ones from Tomkin Press[97], which reflects horizontal approaches from anarchist pedgagodies in higher education, in which students and educators operate side-by-side. (Wilkinson and Ashworth 2025).

We do acknowledge the tensions between anarchist practices and the realities of the higher education systems to which we belong, and the risk of coming across as having a somewhat inauthentic, opportunistic take on anarchism. Admittedly, we operate (and, arguably, succeed) within a system that is not as flexible, self-governing, or non-hierarchical as the one we are taking inspiration from. Still, we believe incremental, practical changes are a possible way of moving towards more inclusive and democratic realities.

Disclaimer

The collaborative, open-ended nature of some steps described here might suit workshops and short courses better than credit-bearing modules with strictly pre-defined learning outcomes. That said, syllabi are often flexible enough to allow for this ad-hoc, collaborative approach (or at least some tweaking).

[97] https://tomkinpress.com/

Accessing the Toolkit

Go to this chapter's companion repository[98]. There, you will find a toolkit for planning your teaching that includes templates for:

- Character Sheet
- Bestiary
- Adventure Map
- Party Guidelines
- Equipment List
- Skills and Spellbook

These are available as `.docx` and `.rtf` documents, but feel free to adapt them to whatever format or platform you want.

Character Building: Presenting Learners and Educators' Profiles

A good first step is to understand everyone's aspirations and previous experiences. At this stage, invite all students and educators (including yourself) to complete a **Character Sheet**[99]. Once everyone has completed theirs, go through a round of introductions. Focus on understanding everyone's strong points, ambitions, and fears.

The Character Sheet could potentially help identify potential topics for class and barriers to be removed.

Example: Character Sheet

- Creation: 15 min
- Discussion: 15 min

Expectations:
Learning how to manage and collaborate in bigger projects using git/GitHub

Previous education and professional experiences:
Self-taught web-development.

(...)

Study Skills Weaknesses:
Might have developed bad habits from self-teaching.

The Anarchist Bit:

Anarchist pedagogies support and are supported by the construction of subjectivities (DeLeon 2012). Feel free to extrapolate the Character Sheet to an exploration of long-term goals, ideals, and a broader sense of identity, even if (perhaps *especially* if) they diverge from immediate, utilitarian aims of the course.

In some cases, however, discourse around subjectivities and course objectives might intersect: students might want to discuss their preferences for sectors they'd wish to work with (for instance: non-profits instead of big tech), or their affinity with open-source initiatives as a matter of citizenship and participation.

[98] https://zenodo.org/records/17702161
[99] The Character Sheet document was designed to help learners understand and share their own experience, needs, expectations, skills, and challenges. It is informed by Student Persona templates provided by the Open University under Creative Commons Attribution-Non-Commercial licenses.

Challenges Ahead: Establishing Learning Outcomes and Syllabus

Use the **Bestiary** template to establish (or remind) learners of the course learning outcomes and syllabus. Discuss what they will learn, giving examples if necessary. Depending on your course's flexibility, outcomes and topics can be decided on the spot. Usually, you will want to choose a set of learning outcomes and lessons beforehand[100].

Example: Bestiary

• Discussion and adjustments: 15 min

Name:
Create GitHub repository for your project
[x] Main Quest **[ ]** Side Quest

Learning Outcomes:
1. Apply best practices to source code management and versioning

Challenge:
Follow all steps from creating a GitHub account to creating a new (empty) repository.

Relevant Equipment and Skills:
GitHub website, GitHub Desktop, Online learning material on GitHub for beginners.

The Anarchist Bit:

Once again, there should be plenty of space for learners to express themselves by suggesting topics for co-creating their learning experience. Such a practice is present in many other progressive pedagogies (for instance, see Freire 1996).

In practical terms, that can be very beneficial to the course. It's not uncommon that students are informed about emerging practices and tools that can enrich the course and learning experience for everyone. As educators might struggle with keeping up-to-date with professional practices, being attentive to students' perspectives can be really helpful.

The Treasure Map: Structuring the Course and Learning Activities

Use the **Adventure Map**[101] to discuss the sequence of learning material and activities, assessment (beasts), and how they relate to learning outcomes. Keep the discussion open so that learners can suggest, based on their expertise, perceived needs, or fears, how to best organise the learning trajectory. To structure their journey, place items from the **Bestiary** into the journey map, associating them with the course's learning outcomes.

[100] Many programmes require educators to achieve specific learning outcomes. If you are the one in charge of keeping the programme or module specification form, a good practice would be keeping it flexible enough so that new, emerging topics can be brought into the syllabus.

[101] The Adventure Map document was designed to help educators and students to set out learning outcomes, syllabi, and assessments. It is informed by material created by the University of Leeds OD&PL.

Example: Adventure Map

• Discussion and adjustments: 30 min

Session	Topics	Main Quest	Sidequest(s)	Learning Outcomes
1	Source-code management	Create GitHub repository for your project	Follow another student's repository	1. Apply best practices to source code management and versioning

The Anarchist Bit:

The collective effort into creating a learning journey that does not follow a blueprint is characteristic of anarchist pedagogies (DeLeon 2012). Still, students and educators might benefit from some structure, in which case it is advisable to outline a learning path. Feel free, however, to continuously discuss, review, and re-negotiate that path.

That discussion could open opportunities for improving and streamlining teaching delivery. You might notice that students might need more (or less) sessions and materials on specific topics. Alternatively, you might consider making additional content available for students who need (or want) it for either keeping up with the group, or extending their knowledge.

P.A.R.T.Y: Organising Groups and Groupwork

As a group (or groups, depending on the number of students), complete the **Party Guidelines** sheet[102], establishing ground rules on how to deal with each other and the group, as well as roles for each participant.

Go over the **Adventure Map**, discussing how to best achieve the challenges (either individually or as pairs or teams). Discuss different strategies for group work; for instance, should everyone work on every aspect of the assignment, or should they focus on their own areas of expertise and interest?

Party Guidelines should help teamwork run smoothly, including agreements for conflict resolution.

[102] The Party Guidelines document was designed to help learners manage their groupwork. They are informed by frameworks from the Institute for Academic Development (University of Edinburgh 2024) and the Learn Higher CETL at the University of Bradford (Learn Higher 2012a, 2012b) under Creative Commons Attribution-Non-Commercial licences.

Example: Party Guidelines

- Discussion and adjustments: 15 min

Topic	Rule
What should someone do if they cannot attend a meeting or complete work on time?	Group will redistribute work according to members' strengths.

The Anarchist Bit:

When deciding rules for working groups, aiming for consensus would be ideal. A quick introduction and effective guide on how to achieve consensus[103] in group meetings was created by workers' co-op of campaigners Seeds for Change, and could be brought into discussion.

Conducting group work can be challenging, particularly when assessment is involved, as students might question their lack of autonomy within a group. It's worth noting, however, that: (a) individual autonomy and freedom is not necessarily the utmost important value in anarchism (see, for instance, Suissa 2006, 14); (b) although the traditional role of group leaders might be harder to justify, certain groups of anarchist values operated under leaders stripped down of authoritarian coercive powers (a good example being pirates as described by Graeber, n.d.).

In practice, students might need to be reminded of how group work allows them to develop more complex outcomes; and even though we are advocates for self-governing, there might be the need for some mediation from educators in case conflicts arise within a group.

A Kind of Magic: Distributing Learning Material

Whether you are using a textbook, slides, PDFs, online videos, or anything else, use the **Skills and Spellbook** to signpost where students can find learning material that is relevant to their learning journey. This can be tailored to individual lessons and activities, or as a general compendium (Warhammer pun not intended), and they could be presented all at once or separately as the learning advances. Again, you could populate it with resources related to students' aspirations.

Example: Skills and Spellbook

Topic:
Source code versioning

Resource Name:
Online learning material on GitHub for beginners

What it is for:
Learning the basics of GitHub for software versioning.

Study Skills Weaknesses:
Might have developed bad habits from self-teaching.

Information and Instructions:
Accessible through GitHub's YouTube playlist[104]

Equipment List: Providing Software and Resources

To complete the setup, provide your group with an **Equipment List** of software and other resources they might need to use to advance the journey. You should have a preliminary list ready, but be open to suggestions from students. Below, an example that would support our student with his desire to learn source code management and versioning.

Example: Equipment List

Websites:
Name: GitHub
Purpose: Source code management and versioning
URL: https://www.github.com

Software:
Name: GitHub Desktop
Purpose: Source code versioning
URL for download or installation instructions:
https://desktop.github.com/download/

103 http://www.seedsforchange.org.uk/shortconsensus

104 http://www.youtube.com/playlist?list=PL0lo9MOBetEFcp4SCWinBdpml9B2U25-f

Setup Complete: Happy Adventures!

Once you have completed the previous steps, you will have generated everything you and your students need to accomplish course objectives and keep track of your achievements. At the start of each learning session, use the **Adventure Map** to update students on their next steps, and on how far they have travelled so far.

Throughout the semester, following the plan from the **Adventure Map**, present new tasks, resources, and skills, from the **Bestiary**, Equipment **List**, and **Skills and Spellbook**, respectively.

Frequently remind students to keep their **Character Sheet** updated with the new skills they acquire.

If conducting group work, remind them of the rules expressed in their **Party Guidelines** whenever there is conflict.

At the end of the course, remind students of their journey, highlighting the learning outcomes achieved along the way. Hopefully, this collaborative approach to learning will have made your job (and of your students) easier and more enjoyable. We would love to hear from you about your experiences, so please feel free to get in touch!

Happy teaching.

The Anarchist Bit:

Another characteristic of anarchist pedagogies, particularly in the arts (and programming is often seen as a liberal art), is how they embrace the fluidity that is necessary for making changes to the learning experience after reflection and evaluation (Wilkinson and Ashworth 2025). After running your campaign, don't be afraid of modifying, discarding, or adapting the system we just proposed according to your experience and that of your students.

Acknowledgement

The authors would like to thank Samantha Ahern and Lucia Michielin for their revision and feedback.

References

DeLeon, Abraham P. 2012. "'Anarchism… Is a Living Force Within Our Life…' Anarchism, Education and Alternative Possibilities." *Educational Studies* 48 (1): 5–11. https://doi.org/10.1080/00131946.2012.641849.

Freire, Paulo. 1996. *Pedagogy of the Oppressed*. Translated by Myra Bergman Ramos. Penguin.

Graeber, David. n.d. *Pirate Enlightenment, or the Real Libertalia*. Penguin Books.

Learn Higher. 2012a. "Ground Rules for Group Work." Association for Learning Development in Higher Education. https://aldinhe.ac.uk/product/learnhigher-resources/ground-rules-for-group-work/.

———. 2012b. "Group Work Booklet." Association for Learning Development in Higher Education. https://aldinhe.ac.uk/product/learnhigher-resources/group-work-booklet/.

Suissa, Judith. 2006. *Anarchism and Education: A Philosophical Perspective*. Routledge. https://doi.org/10.4324/9780203965627.

University of Edinburgh. 2024. "Group Working." *Institute for Academic Development*. https://institute-academic-development.ed.ac.uk/study-hub/learning-resources/group-working.

Wilkinson, Ryan Gerald, and Connor Ashworth. 2025. "Fragments of Anarchism in Higher Education Critical Art Pedagogies." *Arts and Humanities in Higher Education* 24 (3): 259–77. https://doi.org/10.1177/14740222413133033.

Removing Barriers by Programming Without Computers

David Cutting

School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfast
d.cutting@qub.ac.uk

Neil Anderson

School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfast
n.anderson@qub.ac.uk

Matthew Collins

School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfast
m.collins@qub.ac.uk

Introduction

Computational thinking (CT) has increasingly been recognised as a key competence for learners in the 21st century (Wing 2006). It is not limited to computer scientists but has been identified as a "universally applicable attitude and skill set" (Wing 2006). CT involves problem-solving processes that enable individuals to approach complex challenges by decomposing them into smaller parts, designing algorithms, recognising patterns, abstracting irrelevant details, and applying logical reasoning (Aho 2012). These skills are foundational for programming but are also valuable in wider contexts such as mathematics, engineering, social sciences, and everyday problem-solving (Tedre and Denning 2016).

Despite its importance, the teaching and learning of programming and CT is challenging. For many learners, the initial encounter with programming involves exposure to a programming language and development environment. While these tools are essential for producing executable code, they can impose significant barriers to entry. Syntax errors, cryptic error messages, and the abstract nature of code execution often discourage beginners (Robins, Rountree, and Rountree 2003) and many students focus narrowly on the mechanics of a programming language, rather than grasping the underlying concepts of problem-solving and algorithmic thinking (Koulouri, Lauria, and Macredie 2014).

These difficulties are not confined to school-aged learners. Lifelong learners, professionals seeking to re-skill, and students from non-computing disciplines also experience barriers when first attempting to program. Studies have highlighted that learners without prior computing exposure often report low confidence, high anxiety, and a perception that programming is inaccessible or only suitable for "technical people" (Lye and Koh 2014). Such perceptions contribute to underrepresentation of many groups in computing, particularly women, mature learners, and students from non-traditional backgrounds (Sentance and Csizmadia 2016).

Another barrier is the mismatch between learners' expectations and the nature of programming tasks. In formal education, assessments often emphasise correctness of code rather than exploration of ideas. Learners may therefore see programming as a high-stakes activity where failure is punished rather than as an opportunity for experimentation and iterative problem-solving (Watson and Li 2014). This can lead to disengagement, particularly when feedback comes in the form of obscure compiler errors rather than supportive guidance.

Alternative methods for introducing CT have been explored over the last two decades. The "Computer Science Unplugged" movement has shown that unplugged activities—teaching computing concepts without computers—can successfully build intuition and engagement (Bell et al. 2009). By using games, puzzles, and physical activities, learners can experiment with computational structures in a low-stakes environment. Similarly, tangible resources such as cards, tiles, and worksheets have been used to teach sequencing, logic, and algorithms in ways that reduce dependence on technology infrastructure (Curzon et al. 2014). These approaches are especially valuable in contexts where computer access is limited, such as in under-resourced schools or outreach programmes (United Nations Educational, Scientific and Cultural Organization (UNESCO) 2023).

This chapter discusses two approaches to teaching programming and CT without the use of computers. The first is the **ProgBoard**, a printable tool that enables learners to explore sequence, selection, and iteration. The second is a **play-kit for first-year computing students**, developed to foster computational thinking through playful and tangible activities. The chapter then draws connections between these approaches and the wider challenges of accessibility and inclusion in programming education, concluding

with a reflection on the potential of such methods for diverse learner groups.

Unplugged Programming With ProgBoard

The **ProgBoard**, developed as part of the "Think Like a Computer" initiative (Cutting 2025) and available from thinklikeacomputer.org[105], provides learners with a structured way to engage with programming fundamentals in a tangible form. It is a printable teaching tool designed to introduce the three fundamental programming constructs of **sequence**, **selection**, **iteration**, and **variables** all without requiring a computer.

The board itself consists of a grid on which tokens representing instructions can be placed. Problems are posed in the form of challenges, such as navigating a path, following rules, or producing repeated patterns. Learners construct solutions by arranging instruction tokens in order, creating simple algorithms that can be "executed" by following the board step by step.

Sequence

Sequence is taught by requiring learners to arrange steps in a specific order. For example, if the task is to move an object from one side of the board to another, instructions such as "move forward", "turn left", and "pick up" must be placed in the correct sequence. Learners can then manually follow the sequence to check whether the task is achieved. This provides an embodied experience of algorithm design, reinforcing the importance of ordering in programming.

Selection

Selection is introduced by conditional markers. For instance, learners may encounter a task where an object must only be picked up "if it is red". Tokens representing conditional statements are placed on the board, and learners must branch their instructions based on the condition. This tangible representation of branching logic helps learners to understand one of the most challenging concepts in programming: making decisions based on conditions.

Iteration

Iteration is represented by tokens that indicate repetition. For example, a loop marker may allow a sequence of steps to be repeated a specified number of times. Learners may be tasked with drawing a square by repeating a set of instructions four times. Through physically repeating steps, they grasp the efficiency and power of loops compared to writing out repeated instructions.

Variables

Variables are introduced in the form of counters ("keep count of how many...") initially and then as **named variables**, for example "x is three, move forward x squares and each time your counter is in a red square, add one to x". The use of variables in combination with selection for branching logic and iteration allows learners to see how simple algebra aligns with variables in programming and also introduced the concept of state.

Benefits of the ProgBoard

The ProgBoard reduces cognitive load by focusing on concepts rather than syntax. By externalising algorithms into a physical form, learners can see and manipulate their thought processes. Errors become visible as misplaced tokens or illogical sequences, which can be corrected collaboratively. This is in contrast to the intimidating error messages encountered in programming environments.

The tool also supports **collaborative learning**. Students can work in groups, discuss solutions, and challenge each other to design efficient algorithms. Peer explanation reinforces understanding, as learners must articulate the reasoning behind their token placements.

Research on unplugged activities has shown that such approaches are particularly effective for learners with limited prior exposure to computing (Denning 2017). By engaging learners in hands-on tasks, the ProgBoard provides a stepping stone into programming that lowers anxiety and builds confidence.

Figure 25.1: A ProgBoard being used to break the ice with during international programming teaching in China.



[105] https://thinklikeacomputer.org

The Play-Kit Approach for First-Year Students

A second approach to programming without computers has been developed through the design of a **play-kit for incoming first-year university students**. This initiative, derived from the work undertaken at Maynooth University on their primary school CT workbook, aimed to introduce learners to computational thinking through playful, tangible interactions (Anderson et al. 2025) so is therefore widely applicable and not limited to students studying computer science.

The project emerged from the recognition that incoming students often have diverse levels of experience in computing. Some may have extensive programming backgrounds, while others may never have written a line of code. A one-size-fits-all approach risks alienating both groups. The play-kit therefore sought to provide accessible, engaging activities that introduce CT concepts in a non-intimidating way.

Design Principles

The play-kit was co-designed by students and academic staff, drawing on resources originally developed for primary school learners but adapted to suit a university-level audience. Several guiding principles shaped its design:

1. **Accessibility** - reliance on colour was reduced to support learners with colour vision deficiencies.
2. **Visual instructions** - activities emphasised pictorial guidance rather than dense text, reducing language barriers.
3. **Playfulness** - tasks were framed as puzzles or games, encouraging creativity and engagement.
4. **Breadth of CT skills** - activities covered decomposition, algorithms, pattern recognition, logic, representation, and abstraction.

Examples of Activities

- **Bracelet Patterns**: Students extend and complete sequences of beads arranged in patterns, developing skills in pattern recognition, decomposition, and logic.
- **Code Breaker**: A symbolic substitution exercise where learners decode messages using simple ciphers. This fosters skills in representation and abstraction.
- **Escape the Grid**: A maze-based challenge where characters must move according to specified rules. This introduces conditional logic and algorithmic pathfinding.
- **Castles and Wyverns**: A dice-based game where outcomes depend on nested conditions, encouraging learners to apply probabilistic reasoning and selection.
- **Marble Race**: Learners simulate the timing of marbles released into tracks, exploring concurrency, sequencing, and time-based reasoning.

These activities allow learners to explore computational concepts in a playful environment. Unlike programming assignments that require precise syntax, the play-kit enables safe exploration. Mistakes are opportunities for discussion and learning rather than points lost.

Preliminary Evaluation

Early feedback from workshops suggested that the play-kit was effective in lowering barriers for learners who felt intimidated by programming. Students reported increased confidence, greater willingness to collaborate, and enjoyment of the playful format (Anderson et al. 2025). The tangible, hands-on nature of the activities was particularly valued, as it contrasted with the abstractness of code.

Discussion

The two approaches explored in this chapter—the ProgBoard and the play-kit demonstrate the potential of introducing computational thinking (CT) and programming concepts without direct reliance on computers. While they differ in context, design, and target audience, they share a common aim: to make programming more approachable by lowering barriers and foregrounding concepts before syntax. By situating programming in a tangible and playful space, both approaches challenge the assumption that programming must begin with a computer and instead suggest that thinking skills can, and perhaps should, be nurtured in more accessible ways.

One of the most significant contributions of unplugged approaches is the way they reduce **cognitive load** for beginners. Research has consistently shown that novices often struggle with the dual burden of understanding abstract programming concepts and mastering the mechanics of a programming language at the same time (Sweller 1988). For instance, a student may conceptually understand that a loop repeats a set of actions, but may be unable to express this understanding in Python, Java, or C without encountering frustrating syntax errors. By externalising programming logic into physical forms, as seen in the ProgBoard tokens or play-kit puzzles, learners can focus on what the computer is being asked to do, rather than how the instructions are formally expressed. This separation of concerns provides a clearer path to mastery: concepts first, formal languages later.

Equally important is the potential for these activities to support **diverse learner groups**. Traditional programming environments tend to privilege students who are already confident with computers or who have prior experience in coding. This creates inequities, with learners from non-computing backgrounds often left behind. Lifelong learners, interdisciplinary students, and those

with limited digital literacy can find the initial learning curve intimidating, reinforcing the stereotype that programming is only for "tech experts." By contrast, both the ProgBoard and the play-kit offer accessible, low-barrier entry points where learners can engage in computational problem-solving without needing technical fluency. This is especially relevant in outreach contexts, professional development programmes, and in regions where reliable access to computers is limited.

The **affective dimension of learning** should not be overlooked. Fear of failure is a well-documented barrier in programming education, where small mistakes in syntax can lead to discouraging error messages and wasted time (Watson and Li 2014). In unplugged contexts, however, mistakes become opportunities for dialogue and playful exploration. If a sequence of tokens fails to solve a ProgBoard puzzle, learners can quickly rearrange the steps and try again. If a group misinterprets the rules of a play-kit game, the result is not an intimidating compiler error but a chance to reflect, discuss, and iterate. This transformation of errors into constructive learning moments fosters resilience and helps to cultivate a growth mindset around programming.

Furthermore, the **playful nature** of these activities contributes to learner motivation and engagement. Play has long been recognised as a powerful mode of learning (Papert 1980; Whitton 2022). It encourages experimentation, risk-taking, and persistence, qualities that are central to effective problem-solving in computing. By embedding CT tasks into puzzles, games, and tangible challenges, the ProgBoard and play-kit avoid framing programming as a purely technical skill. Instead, they emphasise creativity, exploration, and fun—qualities that can be especially important for engaging students who might not initially see themselves as "coders." For example, the fantasy-themed "Castles and Wyverns" dice game from the play-kit illustrates how algorithmic logic can be learned through an imaginative context, rather than an abstract lecture on conditional statements.

The **transferability of skills** developed in these unplugged settings is also crucial. While the activities do not produce working code, they cultivate habits of mind—such as decomposition, abstraction, and algorithmic reasoning—that can be readily applied in formal programming environments. In this way, unplugged activities function as a **bridge**. Once learners have gained confidence with the underlying concepts, they are better equipped to face the complexity of programming languages and development environments. This scaffolding effect has been linked to improved retention and success rates in introductory programming courses (Luxton-Reilly et al. 2018).

Finally, both approaches point to the **broader potential of programming without computers**. Their application need not be limited to early stages of computer science education. They could be adapted for interdisciplinary teaching, allowing students in fields as varied as biology, business, or the arts to engage with computational thinking in a way that feels relevant to their domain. They could also be used in informal education settings—libraries, community workshops, or online learning platforms—to broaden participation in computing. For educators, unplugged methods provide flexible, low-cost tools that can be adapted to different curricula, learner profiles, and institutional contexts.

Taken together, these observations suggest that unplugged approaches should not be viewed merely as supplementary activities or temporary scaffolds, but as integral components of a broader strategy to democratise computational education. By making programming more tangible, playful, and inclusive, they help dismantle barriers that have historically excluded many learners from engaging with computing. In doing so, they lay the groundwork for more diverse, resilient, and conceptually grounded cohorts of future programmers.

Conclusion

Programming without computers represents an effective and inclusive approach to developing computational thinking skills. By focusing on concepts rather than syntax, learners are able to engage with programming fundamentals in an accessible, playful, and low-stakes environment.

The ProgBoard demonstrates how simple, printable resources can teach sequence, selection, and iteration. The play-kit for first-year students illustrates how playful, tangible activities can introduce decomposition, algorithms, logic, and abstraction in ways that reduce anxiety and increase motivation.

Together, these approaches highlight the value of unplugged activities as a means of lowering barriers, supporting diverse learners, and broadening participation in computing. They serve as stepping stones to computer-based programming, ensuring that learners enter digital environments with confidence and conceptual clarity.

It is concluded that the continued development and evaluation of unplugged programming approaches has significant potential for education at all levels. By reimagining how programming is introduced, educators can make computational thinking accessible to all learners, regardless of background or prior experience.

References

Aho, Alfred V. 2012. “Computation and Computational Thinking.” *The Computer Journal* 55 (7): 832–35. https://doi.org/10.1093/comjnl/bxs074.

Anderson, Neil, Maria Angela Ferrario, Aidan McGowan, Matthew Collins, Jonathan W Browning, Leo Galway, Philip Hanna, David Cutting, and Darryl Stewart. 2025. “Learning to ‘Think’ Through Playful Interactions: A Play-Kit for Incoming First-Year Computing Students.” In *2025 IEEE Global Engineering Education Conference (EDUCON)*, 1–3. IEEE; IEEE. https://doi.org/10.1109/EDUCON62633.2025.11016605.

Bell, Tim, Jason Alexander, Isaac Freeman, and Mick Grimley. 2009. “Computer Science Unplugged: School Students Doing Real Computing Without Computers.” *New Zealand Journal of Applied Computing and Information Technology* 13 (1): 20–29.

Curzon, Paul, Peter W McOwan, Nicola Plant, and Laura R Meagher. 2014. “Introducing Teachers to Computational Thinking Using Unplugged Storytelling.” In *Proceedings of the 9th Workshop in Primary and Secondary Computing Education*, 89–92. ACM. https://doi.org/10.1145/2670757.2670767.

Cutting, D. 2025. “The ProgBoard.” https://thinklikeacomputer.org.

Denning, Peter J. 2017. “Remaining Trouble Spots with Computational Thinking.” *Communications of the ACM* 60 (6): 33–39. https://doi.org/10.1145/2998438.

Koulouri, Theodora, Stanislao Lauria, and Robert D Macredie. 2014. “Teaching Introductory Programming: A Quantitative Evaluation of Different Approaches.” *ACM Transactions on Computing Education (TOCE)* 14 (4): 1–28. https://doi.org/10.1145/2662412.

Luxton-Reilly, Andrew, Simon, Ibrahim Albluwi, Brett A. Becker, M. Giannakos, Amruth N. Kumar, Linda M. Ott, et al. 2018. “Introductory Programming: A Systematic Literature Review.” In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, 55–106. ACM. https://doi.org/10.1145/3293881.3295779.

Lye, Sze Yee, and Joyce Hwee Ling Koh. 2014. “Review on Teaching and Learning of Computational Thinking Through Programming: What Is Next for k-12?” *Computers in Human Behavior* 41: 51–61. https://doi.org/10.1016/J.CHB.2014.09.012.

Papert, Seymour A. 1980. *Mindstorms: Children, Computers, and Powerful Ideas*. Basic books. https://doi.org/10.1007/978-3-0348-5357-6.

Robins, Anthony, Janet Rountree, and Nathan Rountree. 2003. “Learning and Teaching Programming: A Review and Discussion.” *Computer Science Education* 13 (2): 137–72. https://doi.org/10.1076/csed.13.2.137.14200.

Sentance, Sue, and Andrew Csizmadia. 2016. “Computing in the Curriculum: Challenges and Strategies from a Teacher’s Perspective.” *Education and Information Technologies : Official Journal of the IFIP Technical Committee on Education* 22 (2): 469–95. https://doi.org/10.1007/s10639-016-9482-0.

Sweller, John. 1988. “Cognitive Load During Problem Solving: Effects on Learning.” *Cognitive Sciences* 12 (2): 257–85. https://doi.org/10.1207/S15516709COG1202_4.

Tedre, Matti, and Peter J Denning. 2016. “The Long Quest for Computational Thinking.” In *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*, 120–29. ACM. https://doi.org/10.1145/2999541.2999542.

United Nations Educational, Scientific and Cultural Organization (UNESCO). 2023. “Global Education Monitoring Report 2023: Technology in Education – a Tool on Whose Terms?” Paris, France: UNESCO; GEM Report UNESCO. https://doi.org/10.54676/uzqv8501.

Watson, Christopher, and Frederick WB Li. 2014. “Failure Rates in Introductory Programming Revisited.” In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education*, 39–44. ACM Press. https://doi.org/10.1145/2591708.2591749.

Whitton, Nicola. 2022. *Play and Learning in Adulthood: Reimagining Pedagogy and the Politics of Education*. Springer Nature. https://doi.org/10.1007/978-3-031-13975-8.

Wing, Jeannette M. 2006. “Computational Thinking.” *Communications of the ACM* 49 (3): 33–35.

Practical Approaches to Using Sound and Music in Programming Pedagogy

Tom Mudd

Reid School of Music, Edinburgh College of Art, University of Edinburgh
tom.mudd@ed.ac.uk

Yashique Chalil

Reid School of Music, Edinburgh College of Art, University of Edinburgh
ychalil@ed.ac.uk

Matthew Hamilton

Reid School of Music, Edinburgh College of Art, University of Edinburgh
s0674653@ed.ac.uk

Charlotte Desvages

School of Mathematics and Maxwell Institute for the Mathematical Sciences, University of Edinburgh
charlotte.desvages@ed.ac.uk

Mike Taverne

School of Engineering, Physics and Mathematics, Northumbria University
mike.taverne@northumbria.ac.uk

Introduction

This chapter is designed to serve as a brief tour of connections between sound, music, programming and pedagogy. In particular, it is aimed at educators who want to explore sound and music in their own programming pedagogy. The chapter presents motivations for connecting these domains, introduces prior work in this area, and presents a range of (Hamilton et al., n.d.) that can be taken up or adapted to bring sound and music into coding education.

Making code that engages with sound and music can be fun, even with only a very basic understanding of programming concepts, and regardless of any prior music education. There are many ways in which sound and music may connect with programming. For example:

- **composing with code**: using code to write pieces of music,
- **performing with code**: writing code in a performance to make music,
- **creating instruments**: building software tools for personal music making or for others to use,
- **understanding sound**: coding to analyse and explore representations of sound and/or music,
- **data sonification**: representing existing data as sound or music, either in real-time or with historical data,
- **coding responsive sound and music for games** or other interactive programs.

We will first explore motivations for connecting music with programming, followed by a brief overview of the rich existing work in this domain. The set of practical accompanying resources that can serve as starting points into different kinds of music/coding projects are then introduced. The resources themselves are accessible here[106].

Why Combine Programming With Sound and Music?

Music and programming have a rich intertwined history. Simple but effective connections can be found between programming concepts — such as iteration, abstraction, conditionals, loops — and musical concepts such as rhythm, timing, harmony, pattern and so on. Writing a program that generates sound provides an immediate, engaging way to manifest abstract processes. In a pedagogical sense, the sound output is a form of feedback. Programs don't have to be complicated or sophisticated in terms of the coding to be musically satisfying. Students can create sound and music that they can share with friends and relatives, who may not know the first thing about programming but can nonetheless engage with the creative outcomes.

Programming concepts can be playfully explored in musical contexts. For example, an integer array can be rendered as a piano melody, a drum rhythm, or directly as a sound file (see Figure 26.1). This gives an immediate meaning to the output data, motivating students to care about the specific values in the array, the length of the array, the range of data in the array, different ways of reading and writing values to and from the array and so on. Operations on that array take on musical meanings: ordering an might provide a scale; reversing an array reverses musical material; subdividing an array isolates a particular phrase. Meaningful data that students care about can motivate students to find things out for themselves, and provides a context for the kind of creative explorations that are often lacking on

[106] https://github.com/mhamilt/teaching-programming-music-with-code/

introductory programming courses (Sharmin 2021).

Sound and music can also be used to present and experience data in different ways. Data sonification has a long history (Worrall 2019), from giving a constant awareness of something otherwise invisible as with Geiger counters or medical monitoring devices, to hearing emergent behaviours in dynamical systems, providing auditory graphs for the visually impaired (Walker and Mauney 2010), or creative projects that use the data for musical ends (Bulley and Jones 2011; Barrett and Mair 2014).

There are obvious parallels here with coding and visual art. Environments such as Processing (Reas and Fry 2006) connect programming with image and animation, motivating learning and exploration (Reas and Fry 2007, 1). As explored in the following section, there is a similarly rich history of music making in programming pedagogy. To mix music, art, computers and coding therefore continues a long and fruitful tradition (Reichardt 1971; Dreher 2014; Wang 2017), and can serve as a reminder to both students and educators that coding is fundamentally about making things and is therefore a creative act.

Overview of Existing Work

Music and programming have been connected in a wide variety of ways, sometimes through the addition of libraries for integrating musical inputs and outputs with existing languages, and sometimes through the development of specifically designed languages or environments. As this is a very short chapter, it only scratches the surface of many of the interesting work done in this domain, and the focus is on music/code projects with explicit pedagogical dimensions.

Most mainstream programming languages are able to work with sound and music as outputs. This could be achieved by writing data to a file, e.g. an audio file (see 'An Educator's Guide to WAVE Format Files' below) or a MIDI file (a common musical data format that can be used to synthesise sound). The program could also transmit real-time messages to an audio engine that runs separately, e.g. a standalone synthesiser or sampler[107], a digital audio workstation, or frameworks such as SuperDirt[108]. Most mainstream programming languages will have libraries available for these purposes, making it relatively easy to adapt any simple programming exercise into a sonic/musical exercise.

The Scratch programming language[109] is a well-established starting point for young people to engage with coding. While sound and music are not usually the primary focus for students, there is still considerable scope for creativity and experimentation. Brown and Ruthman present a useful range of project types in their *Scratch Music Projects* book (Brown and Ruthmann 2020), from simple theremin-like interactive instruments, through playing simple riffs, up to thinking about loops, musical structures, generative music, and live coding performance. As Scratch is used more generally for creating games, stories or animations, students can be motivated to explore sound and music as one component in wider creative projects.

Sonic Pi[110] is a popular example of a language specifically designed for music. It is a simple but flexible environment that aims to support school-age students to learn to code by making music (Aaron 2016). As with Scratch, the focus on play and having fun with Sonic Pi can foster a more positive attitude towards programming (Petrie 2022), and can provide a starting point for moving from simple programming concepts to more involved topics such as concurrency (Traversaro, Delzanno, and Guerrini 2024). Sound-making programs can start off very simple with basic commands such as `play 60`, but can be built into entire pieces or performances [111].

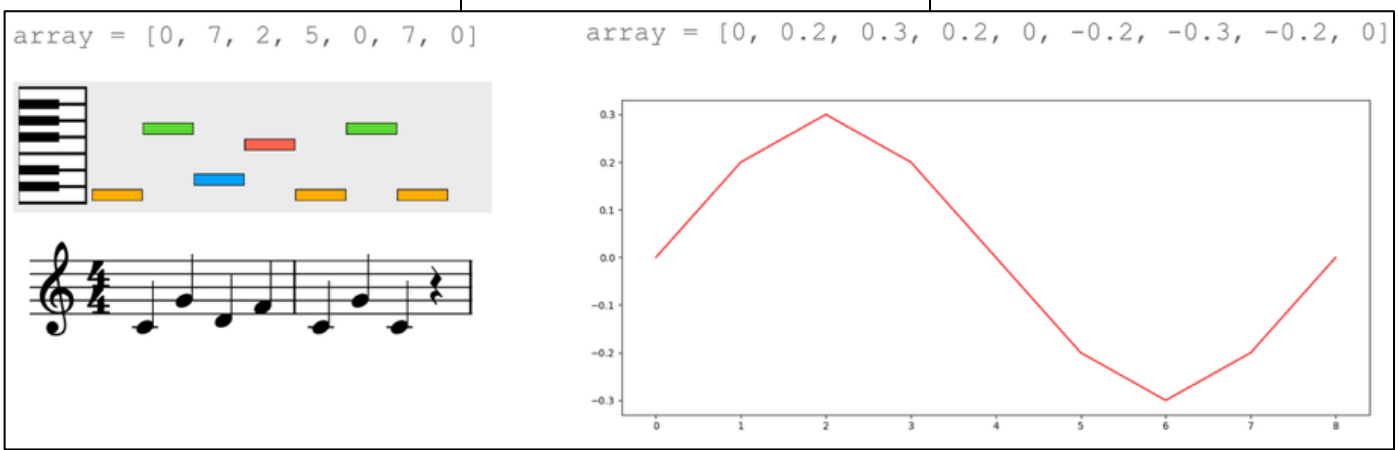


Figure 26.1: Two examples of arrays becoming sound and music. On the left, the array is translated into note values. On the right, the array is translated directly to an audio file.

[107] Lots of open source synthesisers are available such as Surge XT or Dexed.

[108] https://github.com/musikinformatik/SuperDirt

[109] https://scratch.mit.edu/educators/

[110] https://sonic-pi.net

[111] E.g. see work by DJ_Dave in Sonic Pi: https://www.youtube.com/watch?v=w2s1DK1w3WI

Sonic Pi is rooted in practices of live coding for music, a substantial community that serves as an access point to programming for many musicians, as well as an access point to music for some programmers. The movement has engaged closely with programming pedagogy from different perspectives; Blackwell et al (Blackwell et al. 2022) provide a useful overview. The proceedings of the International Conference on Live Coding[112] also present a broader repository of topics, including live coding music in educational contexts (Corvi, Mori, and Nulli 2023; Jara López 2025). Live musical coding is also notable as a community that has attempted to push back against coding as a male-dominated space (Blackwell et al. 2022), with all-women and non-binary workshops being a regular occurrence. Armitage (2018) points to the domain as being a "a space in which to fail constructively".

The EarSketch[113] (Engelman et al. 2017) and TunePad[114] projects are similar to Sonic Pi in their pedagogical aims, in that they seek to broaden participation in computing by demystifying coding and relating it to a domain of interest to students. The projects embed coding elements alongside more conventional music-making tools such as instruments, tracks, a timeline, a mixer, etc. This provides a familiar workspace for those with musical backgrounds and means that the code doesn't need to cover all aspects of the music making, but can be written in snippets with very particular goals, such as making a drum loop, or a bass riff. Both projects run in the browser and use Python (although EarSketch has a JavaScript option). They are designed specifically with sharing in mind, both code sharing and collaborative editing, further motivating students to engage with the coding (Freeman et al. 2019). Petrie (2024) studied the potential for both projects to support computational thinking in 11-12 year old students who had no prior school experience of either programming or music making. Petrie notes that students benefit from the fact that the kinds of musical tasks supported by these projects are naturally incremental and iterative.

A key concern across all the above projects is that there should be room for musical variety and expression; although they may present simple entry points to engaging with programming, it is possible to create rich and interesting creative work that students will be proud of.

Accompanying Resources for Educators

Five example resources accompany this chapter, reflecting some of the different approaches to combining aspects of sound, music and programming in the different authors' existing pedagogical work. Each example resource comes with a `readme` that gives a sense of how the resource might be used, who might find it valuable, what prior knowledge the students may need, and what technologies will be needed. Each resource is described briefly below.

Figure 26.2: Resources available with this chapter.

```
.
├── python-functions-fade
│   ├── functions.ipynb
│   ├── guitar.wav
│   ├── synth.wav
│   ├── piano.wav
│   └── wav_library.py
├── gestural-instruments
│   ├── max/Handshake_rnbo_patch1.maxpat
│   └── web
│       ├── export/
│       ├── img/
│       ├── index.html
│       ├── js
│       │   ├── app.js
│       │   ├── draw.js
│       │   └── guardrails.js
│       ├── ssl/
│       └── style/
├── guide-to-wave-format
│   ├── functions.ipynb
│   ├── ipynb_audio_interactivity.ipynb
│   ├── iteration.ipynb
│   ├── wav_files.ipynb
│   ├── wav_library
│   │   └── c/   c++/   java/   python/   rust/
│   └── wav_library.py
├── python-piano
│   ├── pianopiece_assignment.ipynb
│   └── pianopiece_learning.ipynb
└── strudel-drum-patterns/index.html
```

[112] https://iclc.toplap.org/

[113] https://earsketch.gatech.edu/

[114] https://tunepad.com/

Creating a Piano Piece With Python

This resource is aimed at students who have started on Python quite recently. It attempts to make musical output as simple and accessible as possible. Students are shown how to generate piano pieces as MIDI data which can be played back directly within a Python notebook. Students can develop their understanding of fundamental programming concepts such as lists, loops and conditionals. The ability to directly hear the results is intended to motivate students to think about how they can develop the code by themselves to try to explore how they can start to tailor the musical output towards something they find sonically interesting or satisfying.

The resource is archived as part of the accompanying resources (Hamilton et al., n.d.), and at the time of writing can be accessed here[115].

The following shows a subsection of the Python piano learning resource as a Python notebook, followed by a visualisation of the MIDI output.

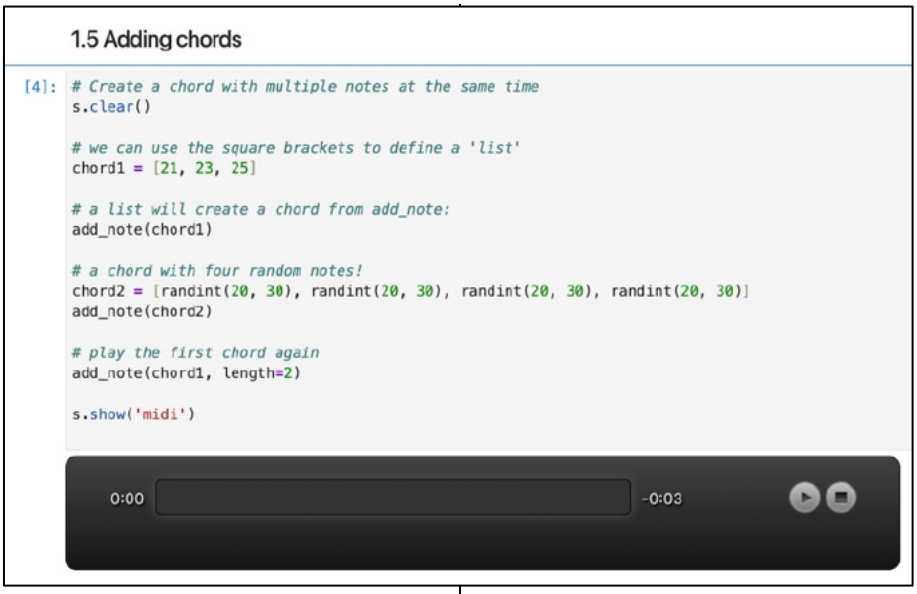


Figure 26.3: Code

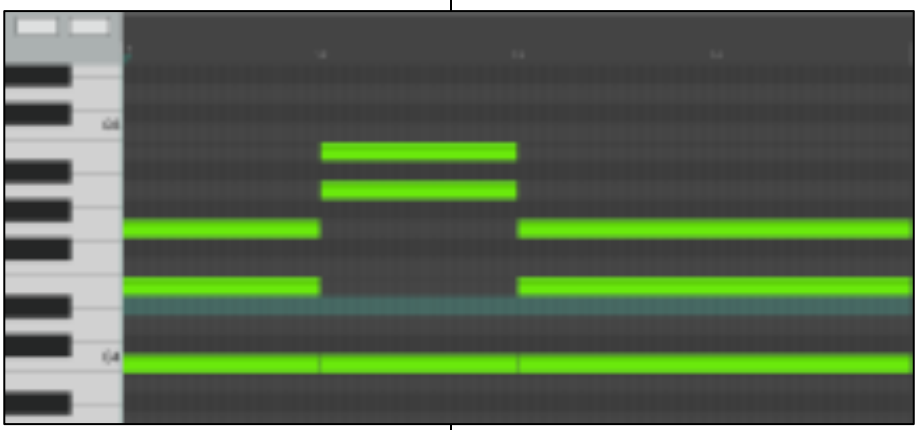


Figure 26.4: MIDI output

115 https://github.com/mhamilt/teaching-programming-music-with-code/tree/main/resources/python-piano

Creating Drum Patterns in the Browser With Strudel

This resource is both similar and different to the piano example. Students engage with Strudel[116], a browser-based JavaScript port of the popular TidalCycles[117] live coding language. The language has been particularly popular for exploring algorithmic approaches to beat making[118]. The editor provides useful visual feedback to show how the code relates to the unfolding beats. Both TidalCycles and Strudel have been popular and accessible ways to engage with code for the first time. They also present a useful introduction to Haskell and functional programming for others. Substantial communities have been built around these technologies (Blackwell et al. 2022), offering routes for those new to either coding or music making to quickly connect to others working in this way.

The resource is archived as part of the accompanying resources (Hamilton et al., n.d.), and at the time of writing can be accessed here[119].

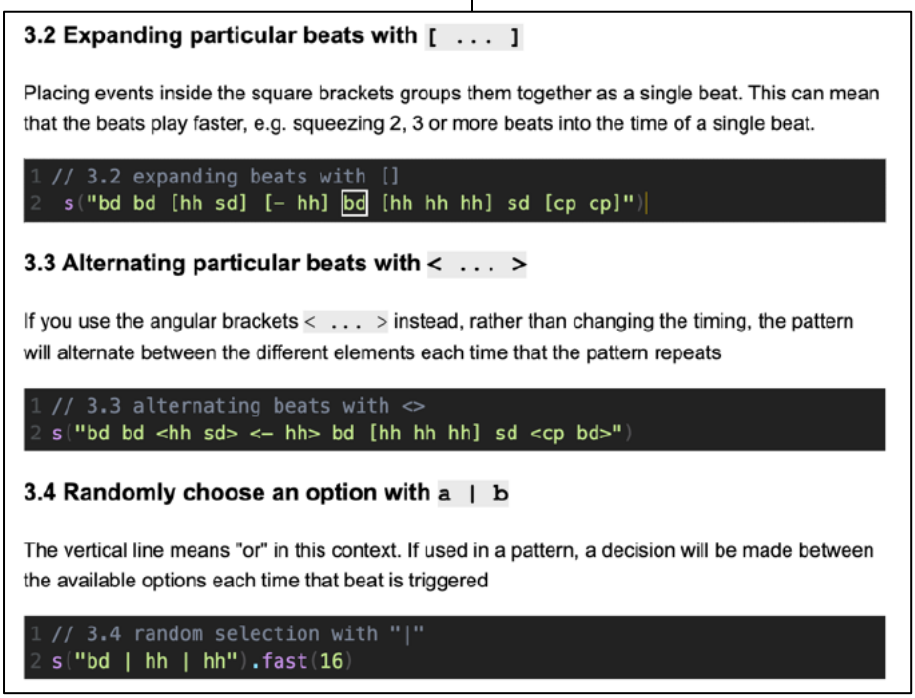


Figure 26.5: Creating beats with Strudel in a browser. The darker boxes can be run in real-time to listen to the results. As the patterns play, the state is reflected visually with the white highlighting that can be seen in the code in section 3.2 of the image.

An Educator's Guide to WAVE Format Files

Resources are contained within the `guide-to-wave-format/` directory (Figure 26.2). The `guide-to-wave-format/wav_files.ipynb` notebook addresses educators directly, laying down the ground work needed to make the portable and technologically accessible audio programming lessons. No prior knowledge is assumed; the resource is intended to guide the reader through the process of WAVE file creation, the ultimate goal being a simple library ripe for customisation and intimately understood for easy dissemination to students.

All data can be represented as sound, and the WAVE file is one of the simplest means of data sonification. Most modern operating systems (and many older ones) will support WAVE file playback through a media player, browser, or command line tool. The resource emphasises the use of standard libraries, both to improve portability and to allow students to get started quickly, instead of burdening them with language-specific ancillary concepts that could potentially be a demotivating obstacle to new programmers.

The intention is not to be prescriptive, but rather show how malleable the WAVE format can be for teaching music and programming.

The central lesson is provided in an IPython (Jupyter) notebook, but the resource in `guide-to-wave-format/wav_library` provides companion code in C, C++, Java and Rust to reinforce the basic elements needed to solve the problem. The resource is archived as part of the accompanying resources (Hamilton et al., n.d.), and at the time of writing can be accessed here[120].

[116] https://strudel.cc/workshop/getting-started/#what-is-strudel
[117] https://tidalcycles.org/
[118] E.g. see the international Algorave movement: https://algorave.com/

[119] https://github.com/mhamilt/teaching-programming-music-with-code/tree/main/resources/strudel-drum-patterns

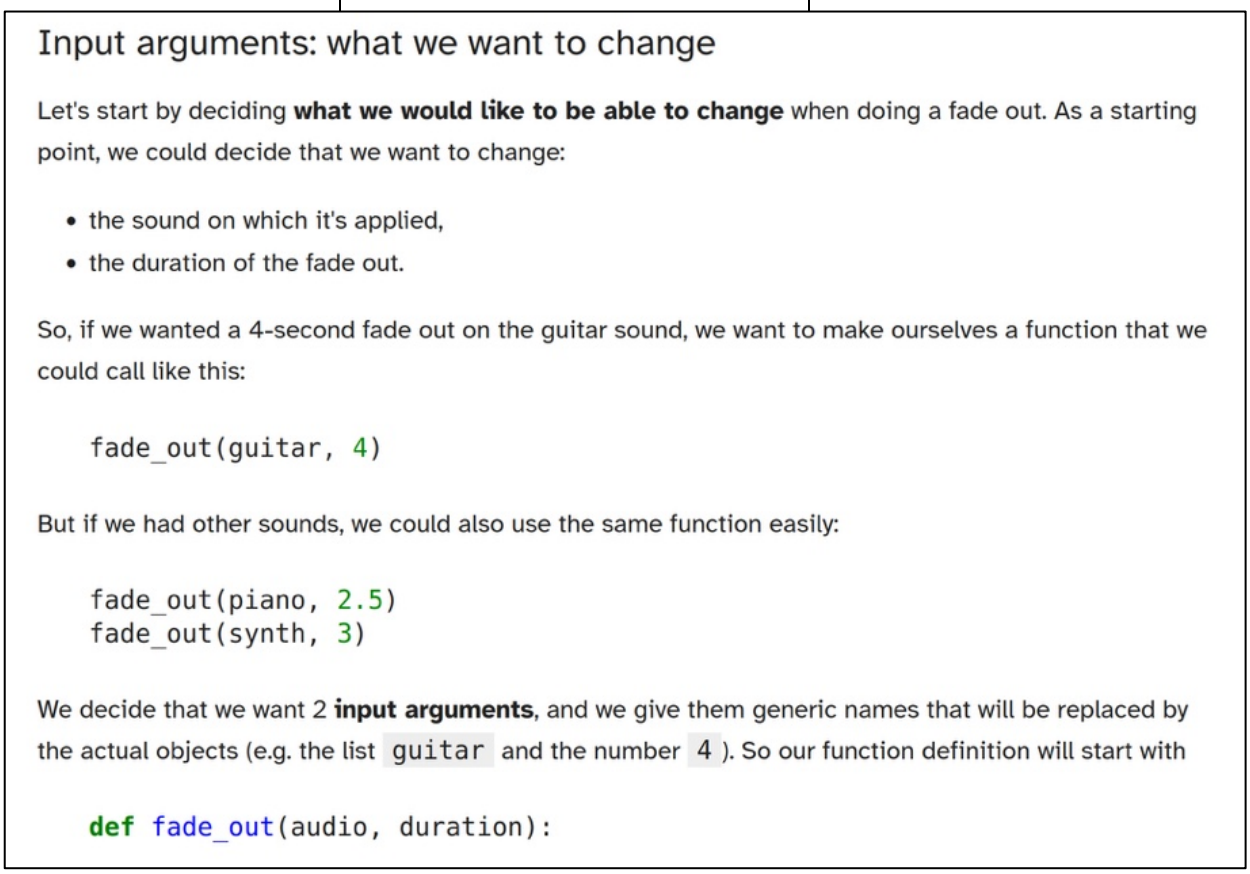
[120] https://github.com/mhamilt/teaching-programming-music-with-code/tree/main/resources/guide-to-wave-format

Introducing Functions by Playing With Audio Files

This resource is aimed at novice Python programmers, with some familiarity with variables, lists, and loops. It motivates the use of user-defined functions and introduces their syntax, using fade-ins and fade-outs on audio files as examples. Throughout the activity, students are encouraged to iterate on their function structure and choice of input arguments and return values, to help them take ownership of their program design, to better understand how and why these choices are made, and to internalise that one-shot perfect code structure isn't necessarily achievable (or even a goal that one should aim for).

Working with audio files is an intuitive way to understand how one chooses input arguments in particular, as the choice naturally arises from "what do I want to be able to control or to change easily", as opposed to "what do I want to hide away and trust that it will always reliably do the same thing" (the function body). The effect of the students' functions is also directly observable (audible, in fact!) by listening to the faded-in or faded-out audio.

The exercises rely on using the Python module `wav_library.py` described below in 'An Educator's Guide to WAVE Format Files' as a tool to read WAVE audio files, with no requirement for students to fully understand it or to have done the activity described there previously. The resource is archived as part of the accompanying resources (Hamilton et al., n.d.), and at the time of writing can be accessed here[121].



Input arguments: what we want to change

Let's start by deciding **what we would like to be able to change** when doing a fade out. As a starting point, we could decide that we want to change:

- the sound on which it's applied,
- the duration of the fade out.

So, if we wanted a 4-second fade out on the guitar sound, we want to make ourselves a function that we could call like this:

```
fade_out(guitar, 4)
```

But if we had other sounds, we could also use the same function easily:

```
fade_out(piano, 2.5)
fade_out(synth, 3)
```

We decide that we want 2 **input arguments**, and we give them generic names that will be replaced by the actual objects (e.g. the list `guitar` and the number `4`). So our function definition will start with

```
def fade_out(audio, duration):
```

Figure 26.6: A short section of the activity notebook, introducing input arguments as a first design choice when writing functions.

[121] https://github.com/mhamilt/teaching-programming-music-with-code/tree/main/resources/python-functions-fade

Gestural Instruments in the Browser

This resource demonstrates the creation of a web-based musical instrument that can be controlled via the sensors on a smart phone. This kind of activity would require students to have some existing experience of the coding language, so would fit well as an exercise or assignment as part of a wider course on music, art, design or programming. The coding is primarily done in Max[122], a proprietary visual programming environment aimed at musicians, as shown in Figure 26.7. RNBO[123] — an extension of Max — allows the visual code to be exported as JavaScript, and will auto-generate a simple HTML page that embeds the project.

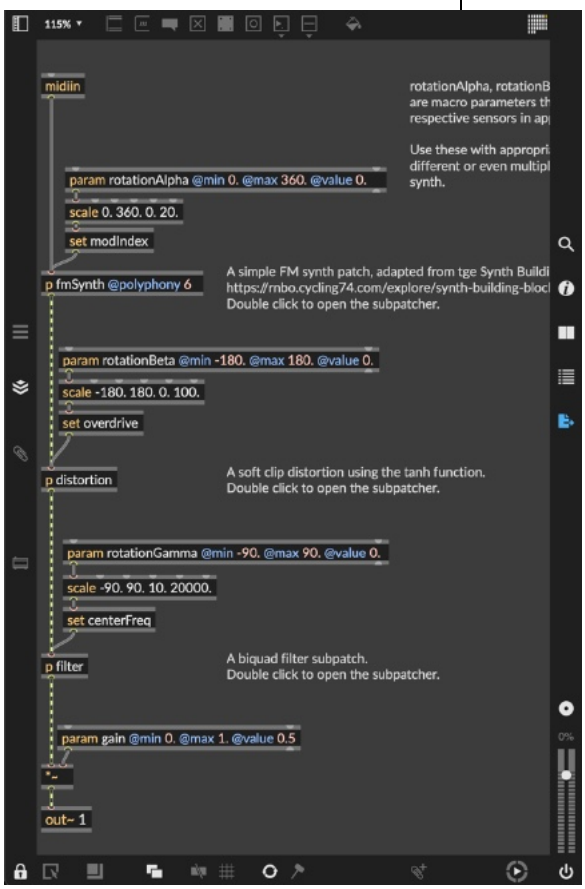
Figure 26.7: Max, a visual programming language is used to create the instrument. RNBO, an extension of MAX can be used to export patches to external targets such as C++, VSTs (Virtual Studio Technology), JavaScript or Raspberry Pi.

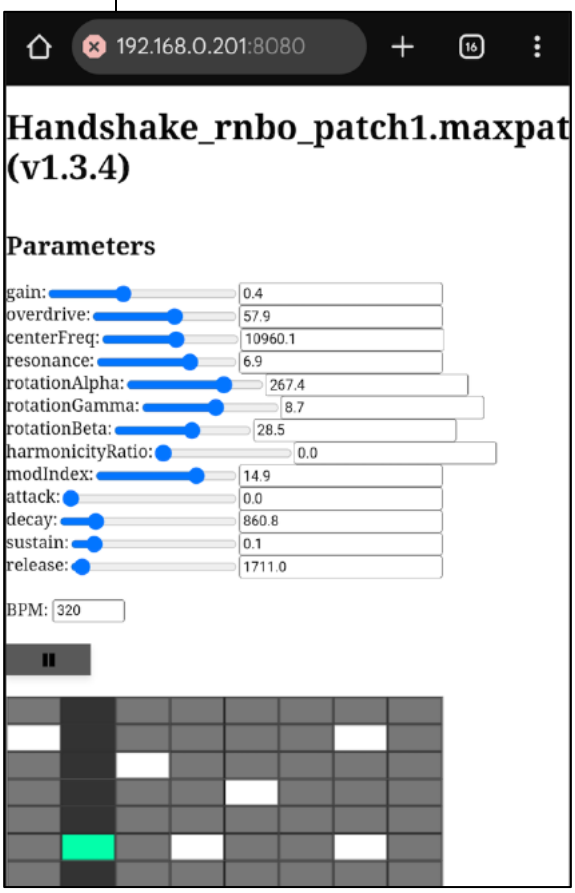
One of the potential benefits of working in this way is that although students start from a visual programming environment, their software can be made available to anyone with a browser (see Figure 26.8), and they begin to get an exposure to JavaScript and web development. As this resource shows, small tweaks to this JavaScript can be added to refine the user interface, and add extra functionality such as smart phone sensor control.

The example provided here is a responsive synthesiser hosted on the web that, when accessed from a smartphone, utilises the motion sensors to control specific parameters of the synthesiser. Once the code has been exported, it can be further developed through the generated JavaScript/HTML. In this example, a simple sequencer for playing notes and visual feedback is added using the p5.js[124] library. Motion data from a smartphone's orientation sensors is mapped in real time so that gestures like tilting or twisting can be used to control the sound.

The resource is provided as a worked example that can be adapted for musical instruments, live installations, audience-participation performances, or experimental audio-visual tools. The resource is archived as part of the accompanying resources (Hamilton et al., n.d.), and at the time of writing can be accessed here[125].

Figure 26.8: Web app exported from Max, which can be accessed on a mobile web browser. The program will then map the relevant motion sensors to the assigned parameters.





[122] https://cycling74.com/products/max
[123] https://cycling74.com/products/rnbo
[124] https://p5js.org/
[125] https://github.com/mhamilt/teaching-programming-music-with-code/tree/main/resources/gestural-instruments

Building a Digital Circuit To Play Sounds and Notes Using Logisim Evolution

This resource explores sound and music making in the context of logic circuits. It is aimed at students working with the basics of digital logic circuit simulator Logisim Evolution[126]. It demonstrates how to use a buzzer to play sounds and notes. The ability to generate sounds is intended to make learning about digital logic more fun and engaging.

Only minimal knowledge is required to test run the project and it can be used as part of a tutorial on digital logic and Logisim Evolution. Alternatively, it could be used as an assignment. It can also be used in music teaching to explain how notes relate to frequencies.

Three main example circuits are provided: a minimal example showing how to use the buzzer; a slightly more advanced example, introducing a bit extender and an adder; and a circuit playing the C major scale using a counter and a multiplexer (Figure 26.9).

Motivated students/learners can then potentially go further by creating a complete digital piano keyboard or synthesizer, or play preset tunes from a digital memory. Such circuits could then potentially be implemented in real hardware later on.

The resource is archived as part of the accompanying resources (Hamilton et al., n.d.), and at the time of writing can be accessed here[127].



Figure 26.9: A digital circuit in Logisim Evolution, used to play the C major scale. The notes are encoded as hexadecimal frequency values and provided as input to a multiplexer. A counter (created as a sub-circuit) runs continuously and is used to select the multiplexer inputs (i.e. the note frequencies) in ascending and descending sequence and pass them on to the buzzer to be played.

[126] https://github.com/logisim-evolution/logisim-evolution

[127] https://github.com/mhamilt/teaching-programming-music-with-code/tree/main/resources/logisim-sound-examples

Summary

This chapter has touched briefly on some of the many connections between sound, music, and programming that exist, and how these can be and have been productively brought to bear on programming pedagogy. We close with Rebecca Fiebrink's advice to musical coders (Brown and Ruthmann 2020, 145), that captures some of the excitement of bringing these domains together:

> Have you figured out how to recreate your favorite pop song with code? Great! Now try composing a new song that's all your own. Or see if you can create new types of music that are easier to make with code than without it — or even types of music that are impossible to create without a computer! What do you come up with? Can you make new sounds that nobody in the world has ever heard before? Can you make musical "instruments" that you can interact with to perform music that you could never make with a piano or a violin? What else could you create, that nobody before you has ever made?

Appendix: Resource List for Educators

Table 26.1: An non-exhaustive list of software and software libraries for exploring music and sound in coding contexts.

Name	Type	URL
EarSketch (Python or Javascript)	Browser IDE	https://earsketch.gatech.edu
gm (R)	Library	https://flujoo.github.io/gm
Mido (Python)	Library	https://mido.readthedocs.io
Max	Visual coding	https://cycling74.com
PureData	Visual coding	https://puredata.info
Scratch	Browser IDE, Visual coding	https://scratch.mit.edu
Sonic Pi (Ruby)	Live coding environment	https://sonic-pi.net
Strudel	Browser IDE	https://strudel.cc
SuperCollider	Programming language	https://supercollider.github.io
Tau5	Browser IDE	https://tau5.live
TidalCycles	Live coding environment	https://tidalcycles.org
TunePad (Python)	Browser IDE	https://tunepad.com

References

Aaron, Sam. 2016. “Sonic Pi – Performance in Education, Technology and Art.” *International Journal of Performance Arts and Digital Media* 12 (2): 171–78. https://doi.org/10.1080/14794713.2016.1227593.

Armitage, Joanne Louise. 2018. “Spaces to Fail in: Negotiating Gender, Community and Technology in Algorave.” *Dancecult: Journal of Electronic Dance Music Culture* 10 (1): 31–45. https://doi.org/10.12801/1947-5403.2018.10.01.02.

Barrett, Natasha, and Karen Mair. 2014. “Aftershock: A Science–Art Collaboration Through Sonification.” *Organised Sound* 19 (1): 4–16. https://doi.org/10.1017/S1355771813000368.

Blackwell, Alan F., Emma Cocker, Geoff Cox, Alex McLean, and Thor Magnusson. 2022. *Live Coding: A User’s Manual*. MIT Press.

Brown, Andrew R., and S. Ruthmann. 2020. *Scratch Music Projects*. Edited by Andrew R Brown and Alex S Ruthman. Oxford University Press.

Bulley, James, and Daniel Jones. 2011. “Variable 4: A Dynamical Composition for Weather Systems.” In *Proceedings of International Computer Music Conference,(ICMC 2011)*. ICMC.

Corvi, Francesco, Giovanni Mori, and Giovanni Nulli. 2023. “Live Coding and Education. A Practical Experience.” In *Proceedings of the 7th International Conference on Live Coding (ICLC) 2023, Utrecht*. https://doi.org/10.5281/zenodo.7843819.

Dreher, Thomas. 2014. *History of Computer Art*. Lulu Press, Inc.

Engelman, Shelly, Brian Magerko, Tom McKlin, Morgan Miller, Doug Edwards, and Jason Freeman. 2017. “Creativity in Authentic STEAM Education with EarSketch.” In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, 183–88. ACM. https://doi.org/10.1145/3017680.3017763.

Freeman, Jason, Brian Magerko, Doug Edwards, Tom Mcklin, Taneisha Lee, and Roxanne Moore. 2019. “EarSketch: Engaging Broad Populations in Computing Through Music.” *Communications of the ACM* 62 (9): 78–85.

Hamilton, Matthew, Tom Mudd, Charlotte Desvages, and mtav. n.d. “Mhamilt/Teaching-Programming-Music-with-Code: Zenodo Integration Test.”

Jara López, Jesús. 2025. “The Live Coding Toolkit. Designing a Methodology for Music Teaching Through Live Coding for Primary Education Students.” Zenodo. https://doi.org/10.5281/zenodo.15528530.

Petrie, Christopher. 2022. “Programming Music with Sonic Pi Promotes Positive Attitudes for Beginners.” *Computers & Education* 179: 104409. https://doi.org/10.1016/j.compedu.2021.104409.

———. 2024. “Design and Use of Domain-Specific Programming Platforms: Interdisciplinary Computational Thinking with EarSketch and TunePad.” *Computer Science Education* 34 (4): 645–78. https://doi.org/10.1080/08993408.2023.2240657.

Reas, Casey, and Ben Fry. 2006. “Processing Code: Programming Within the Context of Visual Art and Design.” In *Aesthetic Computing*, edited by Paul A. Fishwick, 476. MIT Press. https://doi.org/10.7551/mitpress/1135.003.0015.

———. 2007. *Processing: A Programming Handbook for Visual Designers and Artists*. MIT Press.

Reichardt, J. 1971. *Cybernetics, Art and Ideas*. Edited by Jasia Reichardt. New York Graphic Society.

Sharmin, Sadia. 2021. “Creativity in CS1: A Literature Review.” *ACM Transactions on Computing Education* 22 (2). https://doi.org/10.1145/3459995.

Traversaro, Daniele, Giorgio Delzanno, and Giovanna Guerrini. 2024. “‘Hear’ and ‘Play’ Students Misconceptions on Concurrent Programming Using Sonic Pi.” *Informatics in Education* 23 (3): 679–718. https://doi.org/10.15388/infedu.2024.22.

Walker, B. N., and L. M. Mauney. 2010. “Universal Design of Auditory Graphs: A Comparison of Sonification Mappings for Visually Impaired and Sighted Listeners.” *ACM Trans. Access. Comput.* 2 (3). https://doi.org/10.1145/1714458.1714459.

Wang, Ge. 2017. “A History of Programming and Music.” In *The Cambridge Companion to Electronic Music*, edited by Nick Collins and Julio d’Escrivan, Second, 58–74. Cambridge University Press.

Worrall, David. 2019. “Data Sonification: A Prehistory.” In *Sonification Design: From Data to Intelligible Soundfields*, 3–21. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-01497-1_1.

Ways to Teach Online: Lessons Learned Through Experience

Chris Cooling

Early Career Researcher Institute, Imperial College London
c.cooling10@imperial.ac.uk

Samantha Alvarez Madrazo

School of Public Health, Imperial College London
s.alvarez-madrazo@imperial.ac.uk

Joseph El Gemayel

Computer & Information Sciences, University of Strathclyde
joseph.el-gemayel@strath.ac.uk

Nick Jayanth

School of Public Health, Imperial College London
nick.jayanth19@imperial.ac.uk

Lucia Michielin

Edinburgh Futures Institute, University of Edinburgh
lucia.michielin@ed.ac.uk

Leila S. Shafti

Computer & Information Sciences, University of Strathclyde
leila.shafti@strath.ac.uk

Introduction

In this chapter, educators with a background in teaching online will share some of their experiences. These reflections are intended to be informative and to provide insight, inspiration, and advice to other educators who are undertaking similar teaching. Specifically, the chapter discusses:

- A "Silent Disco" at The University of Edinburgh, where students work through material at their own pace with tutors available to facilitate,
- An introductory Python course developed on Coursera for early-career researchers at Imperial College London,
- Four Massive Open Online Courses (MOOCs) integrated into a Masters of Public Health at Imperial College London, including discussions of assessment and interaction between the learners and staff,
- A selection of blended work-based modules delivered at the University of Strathclyde, designed to be taken by learners while they are in employment.

Silent Disco

In this section, Lucia describes her experience in developing "Silent Disco" training activities within the context of the Centre for Data, Culture, & Society (CDCS)[128] *at The University of Edinburgh.*

One of the notable online teaching innovations developed during the pandemic as part of the Training Programme was the "Silent Disco." Conceived as a response to widespread "Zoom fatigue" (Sklar 2020), this format deliberately moved away from the standard model of extended video calls. Instead, it sought to create a learning environment that recognised the variety of ways in which participants approach technical training.

At its core, the Silent Disco is a facilitated, partially asynchronous workshop. Participants join a shared Microsoft Teams chat at a scheduled time, and they work through structured materials at their own pace, rather than listening to a live lecture. These may include coding notebooks, short tutorials, or step-by-step guides. Throughout the session, which normally ranges between two to three hours, instructors are present in the chat to provide support, answer questions, offer clarifications and point to further learning material. The absence of video and audio calls reduces the cognitive load of constant online presence, while the live facilitation ensures that learners are never left entirely on their own (Park, Calistro, and Lipnevich 2025). The possibility of concentrating the focus solely on individual learning experiences was recognised as a particular point of strength by the attendees, as testified by the feedback we received:

> "Silent disco is just an awesome format for learning digital methods, better than live/synchronous webinars because a lot of digital method steps require the learner grappling with the code/steps ourselves. which is better done

[128] https://www.cdcs.ed.ac.uk/

> when alone (in silence)". — Learner feedback

To further support the students, the chat remains open for the duration of the semester following the event. This extended access allows participants to revisit the workshop materials at their own convenience and attempt to apply the tutorials to their own datasets. Moreover, attendees are encouraged to use this platform to share relevant updates, insights, or interesting findings with their peers, fostering a continuing community of learning and collaboration. This ongoing engagement ensures that the Silent Disco remains a dynamic resource beyond the initial workshop session.

The Silent Disco format is particularly well-suited to audiences who may not see themselves as "coders" and have limited time for learning new methods. Based on feedback, mostly from staff, they appreciated the format because they couldn't commit to a full course, but also found complete self-learning challenging. For students and researchers in the arts, humanities, and social sciences, technical content can often feel intimidating, especially when delivered at the pace of a conventional lecture. The Silent Disco allows participants to pause, reread, and experiment without the pressure of keeping up with a live demonstration. At the same time, it provides an immediate channel for assistance, which is often crucial when encountering coding errors or unfamiliar interfaces.

CDCS has applied the Silent Disco approach across a diverse set of workshops. Some have focused on foundational skills, such as database design and SQL querying. Others have introduced intermediate methods for text analysis, sentiment detection, and data visualisation. More creative sessions have included work on explorations of large language models and the usage of regular expressions to correct OCR[129] errors. This range illustrates the flexibility of the Silent Disco model: it accommodates both the practical skills that underpin reproducible research and more experimental engagements with data.

While the Silent Disco format offers a dynamic and flexible approach to learning, it does have certain limitations. For participants who prefer direct, real-time interaction, the absence of live video guidance can make it difficult to deal with complex errors, which could be more efficiently resolved in person or through screen sharing. Additionally, managing multiple asynchronous queries can be challenging for facilitators, potentially leading to delays in addressing individual issues.

What the Silent Disco demonstrates, though, is that online teaching can be both efficient and versatile. By shifting the emphasis away from continuous video interaction, it reduces fatigue while also validating different learning styles. For non-coders in particular, it creates a safe space to approach computational methods incrementally, supported but not overwhelmed. Although born of the pandemic, the model continues to offer a valuable pedagogical alternative, reminding us that inclusive design often begins with the recognition that there is no single "right" way to learn.

"Introduction to Python for Researchers", a Coursera MOOC

In this section, Chris describes his experience developing an introductory Python course on Coursera for the Early Career Researcher Institute (ECRI) at Imperial College London.

Between Summer 2023 and Winter 2024, I developed a course named "Introduction to Python for Researchers" in conjunction with the Interdisciplinary EdTech Lab (IETL)[130] at Imperial College. This course was hosted on Coursera and was based on a previous course I had developed in Blackboard. The course is intended to take 10-15 hours to complete and covers the basics of Python programming. The course is primarily aimed at early career researchers (such as PhD students and postdocs) and is designed to be taken on-demand by learners when they feel it benefits their professional development.

We aimed to distinguish the course from other introductory Python courses by focusing the examples on simplified scientific problems, and best-practice and tools and techniques that would be useful in the real world. In formulating the course, we chose to minimise the number of videos, using images, gifs and embedded runnable code cells in the course pages as much as possible. This is because the time investment to create and maintain videos is much greater than these other media.

The development process took around 18 months where it made up around half of my full-time workload. I developed the content and my colleagues at the IETL formatted it on Coursera, and advised on the pedagogy of online courses. At the time of writing, the course has been live for around 8 months and has had around 1300 enrolments, with 145 completions. We were able to configure the course so that accessing the full version of the course as a member of the public required a paid Coursera subscription, generating some income for us. For students at our university, however, they were able to access an identical duplicate of the course for free using their institution's email address.

Developing this course was hard and time-consuming. The course has 110 pages and contains around 60,000 words. Although the subject was very well-known to me and I had previously produced similar courses to draw inspiration from, creating the content took a long time. There were normal pedagogical considerations relating to

[129] Optical Character Recognition (OCR) is a technology used to extract machine-readable text or characters from raster images or scanned documents.

[130] https://www.imperial.ac.uk/interdisciplinary-ed-tech-lab/

what to cover, how to order it, and what examples and exercises to include. There were also issues relating to Coursera and producing a MOOC specifically — how to render and format the content on the site, how to direct students to forums with questions, and how to use the technology of Coursera to facilitate examples and exercises. Resolving these questions also required the most intensive collaboration with the IETL. In all, developing the content probably took around half the effort, with the other half spent on issues relating to Coursera.

Coursera offered some useful tools for creating the course — one of the most useful being the Coursera Lab. This is a configurable containerised environment with a Visual Studio Code interface. We used this for all of the exercises and some of the examples. For the exercises, we were able to define a suite of unit tests that could check the learner's code to see if it behaved properly. By anticipating some common mistakes a student might make, we were able to include specific feedback messages to help learners correct common mistakes. Coursera also allowed for passing these tests to be required for progress of completion of the course; however, we chose not to do this, and to make only passing the multiple-choice exam at the end of the course a requirement.

One ongoing cost of this course is monitoring. As the main tutor, I check the course's forums once a week for questions and problems. At the current modest user base of the course, questions are relatively rare, but if the course got more popular, this would eventually create a significant workload. On one occasion, the way Coursera implemented the Labs behind the scenes unexpectedly changed and this broke the grading in all of our labs. This led to many questions and lots of feedback from struggling learners. I was grateful that my colleagues at the IETL were able to quickly identify and fix the problem, but this was an unexpected and significant amount of work. It seems possible this could occur again.

Overall, internal and external students both seem happy with the course. For our internal students I think there is a slight improvement in experience compared to our old Blackboard course that this course replaced. A downside is that developing the Coursera course did take more time and effort as the interface is more fully-featured and complex. Commercially, this course has not been a significant success. The area of "introductory Python course" is very saturated, making it difficult to become very successful.

If I were developing a Coursera course again, one thing I would do is learn how to do more of the formatting myself. I drafted the content in one place, the IETL formatted it on Coursera, and then I checked it and we iterated if necessary. Whilst my IETL colleagues were very competent and helpful, this process introduced significant extra work. If I had produced most of the content directly on Coursera, leaving only the more complex bits for my more experienced colleagues, it would have significantly reduced the overall amount of work.

Overall, I'm pleased to have developed the Coursera course. It provides a great asynchronous resource for learners to study the basics of Python. As this is one of our most popular topics, it supports a large number of students and frees up a lot of teaching time, helping to justify the large time and effort outlay of creating the course. Careful consideration of the content, sequencing, problems users might encounter, formatting and which examples and exercises would be most useful help make it a useful resource for learners.

Comparing MOOCs and Online Degree Content

In this section, Samantha and Nick compare how content was created and is currently delivered in four MOOCs for Statistics for Public Health and the corresponding Statistics for Public Health core module (previously known as specialisation), which is part of the Master of Public Health online (MPHO) degree at Imperial College London.

Massive Online Open Courses (MOOCs) are online courses, not often associated with any formal qualifications. There is no cap on the number of students who can take these courses. Although the content could be developed by highly qualified tutors in the field, they would have little to no interaction with the students who are enrolled, and the students would need to rely more on the course content. In comparison, online degree content is content associated with a formal high-level qualification such as a certificate, diploma, or a degree. There would be similar structures to in-person degrees in place, such as an admissions process, a cap of student numbers, and two-way interactions between students and tutors. (School 2021).

Four MOOCs for Statistics in Public Health were developed in Coursera by Prof. Alex Bottle, Prof. Victoria Cornelius, and Dr. Lisa Danquah from the academic team, and Dr. Argita Zalli and Helen McKenna from what was at that time the Digital Learning Hub at Imperial College London. Each MOOC consisted of four weeks with videos and readings, some which focused on statistical concepts and others on coding in R. There were also quizzes included to reinforce concepts being learned. Each MOOC has a specific dataset, which allows students to learn statistical concepts and the corresponding R code. The MOOCs were launched in Coursera between November and December 2018. Samantha started taking care of the

delivery in 2020 and Nick joined in 2023.

The MOOCs were developed for several purposes. Two important ones were to be aligned to Imperial's strategy of creating worldwide collaboration to meet global grand challenges, as the MOOCs are aimed at anyone who has an interest in learning Statistics for Public Health, and to give the learners a taste of the content in our MPHO programme. If they liked the MOOCs, they have the option to enrol and obtain a Postgraduate Certificate (1 year part-time), Diploma (2 year part-time) or the Master's degree (2 or 3 years part-time). This is well captured by several five-star reviews in Coursera. Here is an example from January 2021:

> "This is a comprehensive and well-made overview of statistical principles and techniques (1) in the context of public health and (2) that will be useful in the subsequent courses in the Coursera Specialization where it belongs. While there are a lot of similar MOOC offerings around, the public health examples and the unique approach this course provides make it worth taking especially if you are the type of person who wants to"cover all bases." This is highly recommended for those aiming to have a career in public health-related research or even those casual learners who want to make sense of the data that they see and hear from the news" — Coursera review

To access the MOOCs, there were originally two options. The first option is to obtain a certificate when paying for full access; this option is still available. This is more suitable for those students who are interested in accessing all the content asynchronously. Each MOOC was designed with four weeks of content. However, Coursera allows other timeframes to complete them. From our understanding, it is not necessary to complete MOOC 1 to move to MOOC 2 and so on.

The second option to access the MOOCs, which is the one we were suggesting to our students holding an offer for the MPHO, was to audit the whole MOOC 1 for free. In that way they could start to get familiar with the statistical concepts, follow instructions to install R/RStudio and start practising in R. However, in August 2025, Coursera replaced the auditing by a 'preview mode' which currently only allows full access to the first week of some of the MOOCs.

Table 27.1 below shows a summary of the number of students enrolled and the completion rate up to August 2025 for the four MOOCs.

For the degree students, there have been several additions to the content. We will provide here three pivotal ones: the different options to increase student-faculty interactions, the mock assessment, and the summative assessments.

Table 27.1: Summary of Students Enrolled on the Four MOOCs and the Completion Rate. Source: Coursera

MOOC No	Title	Enrolled	Completed (%)
1	Introduction to Statistics & Data Analysis in Public Health	60,185	10,962 (18%)
2	Linear Regression in R for Public Health	17,223	4,572 (27%)
3	Logistic Regression in R for Public Health	14,308	3,167 (22%)
4	Survival Analysis in R for Public Health	16,007	3,096 (19%)

Interactions Between Students/Learners and Faculty in MOOCs and Degree Content

There is currently no consensus on the meaning of 'engagement'. However, it has been proposed that one key engagement dimension from the National Survey of Student Engagement is student and faculty interaction (Robinson and Hullinger 2008). This dimension is clearly different between the MOOC learners and the degree students. The MOOC learners have required over the years minimal interactions with staff, and have mainly contacted us when there is an issue to be awarded a certificate. Prof. Bottle monitored the course for a brief period after the MOOCs were launched, but there has not been any reason to continue doing it. Therefore, we have minimal monitoring and modifications to the content. The higher numbers of students have also posed challenges for one-on-one engagement.

In contrast, the degree students have more frequent interactions with staff. These interactions include two induction sessions to support installation of R/RStudio (60 min each), then, during term time, 15 weekly office hours to solve statistical questions (60 min each), 12 weekly R drop-in sessions to solve coding questions (60 min each), forums to post questions related to statistical or coding questions, 13 weekly live sessions to reinforce the online learning (60 min each) and communication via e-mail. The number of degree students enrolled in the Statistics for Public Health module has varied between 2019 and 2025 from 60 to 110. As students are studying part-time and are based worldwide, it is their choice to interact with faculty members or not. Therefore, usually 10-20% have at least one student-faculty interaction. Several of these interactions have been useful to update the degree content (e.g. providing alternative commands to displaying R output in the format provided by newer versions).

Mock Assessment

Compared to the MOOCs, MPHO students have the Statistics module with 12 weeks of compulsory content. This is the standard length of modules in the online degree to allow students time to prepare for their summative assessments. The content from the degree had modifications from the four MOOCs to cover all the essential concepts in 12 weeks. In week 13, MPHO students are provided with an optional formative mock assessment, which is not accessible to the students enrolled on the MOOCs. This is because the mock assessment, which has its own dataset, is directly linked to the summative assessment that MPHO students have to successfully complete in order to pass the module and obtain their high-level qualification. With access to the mock assessment, students can practise the skills acquired throughout the module and write an abstract with the results obtained. From the abstracts submitted each year, two of them are selected as examples and feedback is provided by staff and peers in the last online live session of the module. This allows the students to have a better understanding of how to put together their abstract for the final summative assessment. For the students whose mock assessment abstracts are chosen for the online live session, their abstracts are anonymised and the students receive detailed feedback, which is an incentive to take part.

Summative Assessments

As mentioned earlier, in the paid version in Coursera, a certificate can be obtained from each of the four MOOCs, and to do so, different assessments have to be completed:

- For MOOC 1, seven short quizzes have to be completed (each 10%, with 3 to 6 autograded questions) and then an end-of-course quiz (30%, with 15 autograded questions).
- For MOOC 2, there are two autograded quizzes, both with 7 questions each.
- For MOOC 3, there are 5 autograded quizzes, having between 4 and 10 questions each.
- For MOOC 4, there are 4 autograded quizzes, having between 1 and 10 questions each.

The content of the assessments has remained the same since 2018. However, the content of summative assessments in the MPHO has evolved over the years. This usually takes several days to create. The first cohort had 3 autograded summative multiple-choice questions plus the main assessment, consisting of coding the best model in R to answer a research question with the dataset provided, and writing the corresponding 350-word abstract. The variety of models generated for the abstract created a challenge in terms of marking all the assessments in a standardised manner. Further, despite requiring the students to upload a data flow diagram, it was sometimes hard and time-consuming to work out what they had done, and where they had gone wrong.

Therefore, since the second cohort, the faculty team decided to generate the R code and output themselves. Students have to select, from the R output, the most appropriate model to answer the research question, write an abstract, answer 10 autograded yes/no questions on whether various parts of the output are useful, and finally answer five or six free-text questions, which mainly look at interpreting various outputs.

Recently, owing to Imperial's initiative of reducing assessment load across the University, and the rising risk of AI usage on summative assessments, we amended the summative assessment on the module. From the academic year 2025/2026, there is one summative assessment, which includes one autograded summative multiple choice questions related to the all learned content, plus interpreting and selecting the most appropriate model from the R code to answer a research question, write an abstract and answer five to six free-text questions. Currently the use of AI is not allowed in the summative assessment, hence we have followed Imperial's guidances whilst creating the assessment, to try to make it as AI-proof as possible by including some reflective free-text questions and/or questions to interpret graphs.

These changes in summative assessment have also required over the years updating a few items in the degree content, to ensure we are providing good scaffolding for students, not only to learn statistical concepts and coding in R, but also to prepare them for their summative assessments, specifically their research project where they will employ the skills learnt on the Statistics module to effectively write an abstract. The specific changes to degree content have included providing mock multiple choice questions and two readings, one with FAQs and another one describing in detail what is required in each section in the abstract, with a couple of examples of abstracts from the literature with feedback on how each can be improved according to the marking rubric.

Overall, the use of shared high quality content for both MOOC and degree learners based worldwide proved to be a successful approach. It allows flexible, accessible education to equip learners with basic statistical methods used in the public health field. Those who are interested in specific topics with minimal faculty interactions can take the MOOCs; future work will focus on increasing the amount of free content available and ensuring there are opportunities to keep the MOOCs content up to date. Those who are interested in obtaining a high-level qualification from the School of Public Health at Imperial College London have access to continuously updated content, with increased student-faculty interaction, and scaffolding for the summative assessment which at the same time empowers them to write their own abstracts.

Blended Work-Based Learning

In this section, Joseph and Leila describe their experiences of developing and delivering online modules for the Graduate Apprenticeship and Degree Apprenticeship (GADA) BSc (Hons) programmes in the Department of Computer and Information Sciences at the University of Strathclyde.

Graduate Apprenticeship and Degree Apprenticeship (GADA) programmes are delivered over 4 years with three terms each year. They are blended work-based learning programmes, designed in partnership with industry, to allow the learners to gain a degree while in employment. GADA programmes combine largely asynchronous online learning with one day every 4 weeks of mandatory on-campus activities and one day per week of non-mandatory on-campus activities.

Since 2019, we have taught various modules in these programmes. Our blended model consists of short video lectures, screen-captures of worked examples, directed readings from textbooks, quizzes to self-evaluate learning, hands-on programming exercises, moderated online discussion forums, and on-campus activities. Both authors have contributed to the development of online materials as well as the delivery of the modules.

Before preparing online materials, staff members are offered training by the university to gain skills and experience with recording and editing. During the recording phase, each 10-minute video took approximately one hour to prepare and produce. Each week of a 20-credit module typically includes 10 to 15 videos, and the module runs over 12 weeks. Support from the university team helped with video editing, managing the Moodle page, and handling administrative tasks.

Based on our experiences of teaching these blended modules, we observed that different learning resources play complementary roles in supporting students' learning. Short videos and worked examples provide concise, targeted explanations of specific techniques or programming concepts, while textbooks offer deeper theoretical context. Practical exercises and formative quizzes enable students to test and consolidate their understanding. Online discussion forums further support learning by facilitating peer interaction, which is particularly valuable for GADA students who are balancing academic study with professional responsibilities.

The asynchronous nature of the learning materials is particularly beneficial for apprentices. Students frequently revisit video lectures and worked examples, especially in preparation for assessments. The ability to pause, reflect, and re-watch explanations allows learners to study at their own pace and accommodate the constraints of full-time employment.

Our experience also suggests that participation in online discussion forums increases when activities are clearly structured and linked to specific exercises or tasks. When prompts are broad or loosely defined, students may be unsure how to contribute, and engagement tends to remain limited. Providing clear instructions or discussion prompts helps to establish expectations and encourages more meaningful interaction.

Another important consideration is the maintenance of online content. While the creation of digital materials is initially resource-intensive, ongoing revision is equally important. In some cases, videos must be re-recorded to maintain clarity or reflect changes in software tools and programming environments. This need for regular updates is particularly relevant in computer science, where technologies evolve rapidly.

Ultimately, effective teaching depends on the ability to engage students and foster their enjoyment of learning. When learners are actively engaged, they tend to become more independent and motivated in their studies. Creating such engagement, however, remains one of the key challenges of online and blended learning environments.

Drawing on our experiences of designing and delivering these modules, we offer several practical recommendations for educators developing similar programmes.

Designing Coherent Learning Pathways

Students benefit when the relationships between videos, readings, exercises, and discussion forums are made explicit. When these elements are clearly connected, learners can better understand how each activity contributes to their development. Structuring materials in smaller units is particularly helpful. Dividing longer lectures into short videos focused on individual topics enables a more balanced learning experience, allowing students to alternate between watching videos, reading supporting materials, and completing exercises.

Keeping Video Content Concise and Interactive

Video lectures are most effective when they remain brief and focused. Unlike traditional face-to-face lectures, online videos do not need to repeat explanations in multiple ways within a single session. Students can pause, replay segments, or consult additional resources as needed. Incorporating interactive elements within videos can further enhance engagement, particularly when introducing complex programming concepts. For example, instructors can present a short exercise during the video and encourage students to pause and attempt a solution before reviewing the worked example. Tools such as H5P[131] allow such interactions to be embedded directly into the video environment.

Supporting Discussion and Collaboration

Online discussion forums can play an important role in supporting peer learning, but they require careful design to function effectively. Broad instructions often lead to minimal

[131] https://h5p.org/

engagement, as students may not know how to begin contributing. More focused prompts or discussion tasks linked to exercises or assessments tend to generate more meaningful participation. In addition, learners frequently expect instructors to remain visible in these discussions, even when the primary aim is to encourage peer-to-peer interaction.

Collaborative activities conducted outside the classroom can also enhance engagement. Off-campus group work allows students to share perspectives and experiences from their workplaces, enriching the learning process while maintaining the flexibility required by work-based learners.

Encouraging Practice and Self-Assessment

Programming modules require frequent opportunities for practice. However, too many summative assessments can place considerable pressure on students who are already managing professional responsibilities. For this reason, formative exercises are particularly valuable, allowing students to experiment, make mistakes, and develop their skills without the pressure of grading.

Self-assessment activities also support learning by enabling students to evaluate their own work independently. This process helps build confidence and encourages learners to take greater responsibility for monitoring their own progress.

Providing Timely Feedback

Immediate feedback is an important motivator for learners. When students receive rapid responses to their submissions, they are more likely to correct mistakes and refine their understanding. Automated assessment tools such as Coderunner (Lobb and Harlow 2016) are particularly effective in programming modules, as they allow students to receive instant feedback on their code while reducing the marking workload for instructors. Shafti and El Gemayel (2026) in this volume address the topic of automated marking in more detail.

Maintaining and Improving Learning Materials

Finally, online learning materials should be reviewed and updated regularly, ideally on an annual basis. Designing content in a modular format helps make this process manageable. Short videos focusing on individual topics are easier to revise or replace without requiring the re-recording of entire lectures. Monitoring which resources students revisit most frequently can also provide valuable insights into where students encounter difficulties or require additional clarification. These insights can guide improvements to explanations, examples, and exercises in subsequent iterations of the module.

Overall, our blended-learning approach has proven highly effective for working professionals. Thoughtfully designed, developed, and maintained, it delivers both flexibility and depth, supporting sustained learning over time.

Our work with the GADA programme was acknowledged internally in 2022 when the Strathclyde GADA teaching team received the Teaching Excellence Awards Team Award, reflecting the collaborative effort and effectiveness of our blended learning approach.

Conclusion

In this chapter, we have presented several examples of online teaching. We have had different experiences, which we hope you have found useful and informative, but have also agreed on a few key points.

Online teaching excels at providing learners the ability to learn asynchronously and at their own pace. It is also the only practical way to reach large numbers of learners. However, initially creating the required materials is a big commitment, and a course requires constant monitoring and frequent updates. Videos are particularly resource-intensive and using approaches such as minimising the number of videos, or making more, shorter videos can reduce the workload in creating and maintaining content. Student engagement is encouraged through frequent exercises and tools such as automated feedback help, to make the exercises as useful and engaging as possible. For courses with more learners, automated feedback quickly becomes the only practical method of feedback.

Overall, once an online course has been created, students often find them an effective way to learn. For educators, it can provide a time-efficient way to teach, especially if the course lasts for many years.

References

Lobb, Richard, and Jenny Harlow. 2016. “Coderunner: A Tool for Assessing Computer Programming Skills.” *ACM Inroads* 7 (1): 114–18. https://doi.org/10.1145/2810041.

Park, M. J., T. Calistro, and A. A. Lipnevich. 2025. “Zooming to Work or Working to Zoom: Relations Between Zoom Fatigue, Cognitive Load, and Boredom.” *Research Papers in Education* 40 (6): 822–50. https://doi.org/10.1080/02671522.2025.2507592.

Robinson, Chin Choo, and Hallett Hullinger. 2008. “New Benchmarks in Higher Education: Student Engagement in Online Learning.” *Journal of Education for Business* 84 (2): 101–9. https://doi.org/10.3200/JOEB.84.2.101-109.

School, EDHEC Business. 2021. “Online Diplomas Versus MOOCs: Advantages and Downsides.” https://online.edhec.edu/en/blog/online-diplomas-versus-moocs/.

Shafti, Leila S., and Joseph El Gemayel. 2026. “Using Automated Marking in Programming Courses to Enhance Learning.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Sklar, Julia. 2020. “’Zoom Fatigue’ Is Taxing the Brain. Here’s Why That Happens.” https://www.nationalgeographic.com/science/article/coronavirus-zoom-fatigue-is-taxing-the-brain-here-is-why-that-happens.

Seeing Before Coding, Doodling Before Doing: How Teaching Data Visualisation Transforms the Way Students Think

Suzanne Goopy

Edinburgh Medical School, University of Edinburgh
sgoopy@ed.ac.uk

The Unexpected Course

The scene is set: In a dining room in China, at a kitchen table in Scotland, in a library in Canada, students boot up their laptops, turn on their devices, and come online. In a seminar room in Edinburgh, students file in and take their seats, computers at the ready. Most are postgraduate students — many of them coming from quantitative, scientific, or health backgrounds across the University of Edinburgh. They are individuals interested in and comfortable with numbers, analysis, and programming. They have signed up for a course called *Data Visualisation: Knowledge Transfer*: most have not read the fine print, and they come expecting to write code.

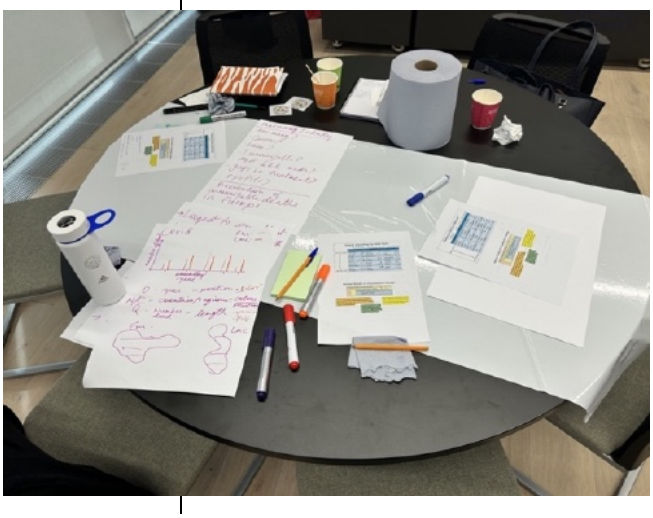
More often than not our students arrive expecting to learn about new chart types, to take a deep dive into *Python* or *R* visualisation libraries, or to discover some other innovative programming activity involving dashboard design. To their surprise, and sometimes horror, they are presented with a pen, a pencil, a piece of paper, and a whiteboard (or the online equivalent). This is not an accident, a resource shortage, or a pedagogical quirk. It is the entire point. From the very first session, the course does something radical as it asks technically capable postgraduate students to put down their keyboards and pick up a pen. It asks them to slow down, to think, to doodle, to draw, to sketch. It asks them to become, possibly for the first time in their postgraduate careers, deliberately and productively uncomfortable and uncertain and, ultimately, it changes forever the way they see data.

This chapter presents a story of transformation by offering some insights into what the course *Data Visualisation: Knowledge Transfer* does, how it does it, and why, at a moment in time when generative AI can produce a bar chart from a prompt in seconds, it matters more than ever that our students can see before coding and doodle before doing.

The Problem With Jumping Straight to Code

There is a seductive logic to the idea that learning data visualisation means learning to code visualisation. After all, programming tools are powerful, expressive, and fast. A student who knows how to use a *Python* or *R* visualisation package can produce a scatter plot in minutes; they can iterate, tweak, re-render, and they can handle thousands of data points with ease. So why do we not start here? Indeed, why do we not go here at all? The answer is that programming tools, for all their power, are also 'cognitive cages'. From the moment a student opens a coding environment, they are limited. Constrained by the chart types the tool offers, the parameters it accepts, the

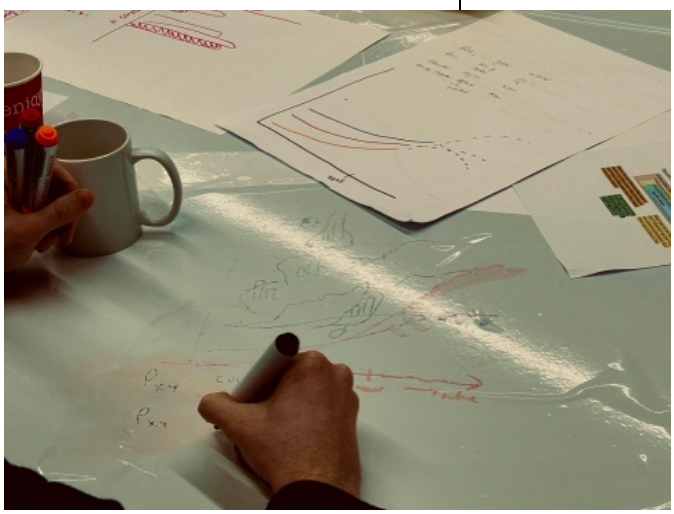
Figure 28.1: To think, to see, to question...



defaults it imposes. The tool shapes their thought. A student reaching for a data visualisation tool will, almost instinctively, reach for a line chart or a bar chart not because these are the right choices for the data or the audience, but because these are what the tool offers first. With this, the creative space collapses before it has even had a chance to open.

This is not the only problem, though, as beyond the constraints of the tools, there is a deeper issue at play. An assumption that visualisation is primarily a technical act rather than a creative and communicative one. When students think about making a chart, they are often thinking about rendering data correctly. They are not thinking about who will read it, what that reader already knows, what question is being answered, what knowledge they want to uncover, what information they want to impart and, least of all, what feeling might be evoked, or what story might be being told. These questions are not peripheral to data visualisation. Indeed, quite the contrary, they are its heart and as such, the course begins from the position that you cannot become a truly skilled data visualiser by learning programming alone; rather you must first learn to see and think … and doodle.

Figure 28.2:
…and to doodle.



The Power of Pen and Paper: Doodling and Sketching as Cognitive Liberation

The introduction of ‘pen and paper’ as a core component of this course is its most distinctive and powerful feature. It comes as a genuine shock to many students. Sometimes there is even visible discomfort with students expressing that this is not quite what they signed up for, or that the analogue medium is not only unexpected but perhaps even somehow beneath the sophistication they have spent years developing. There is discomfort on display, the ‘niggle’ is at work … and growth is on the horizon.

When you doodle, draw, sketch, you are freed from the tyranny of precision for a pencil line does not require you to specify axis labels, font sizes, or colour hex codes; it asks only that you think about shape, about relationship, about proportion and flow. It invites you to add embellishments, indeed, to add whatever you like. The rawness of a hand-drawn sketch is not a deficiency; it is a feature. It signals that this is an idea, not a product, and it invites iteration and critique in a way that a ‘polished’ visualisation does not.

In practice, the course builds ‘pen and paper’ work into every stage of the design process. Students are asked to sketch multiple candidate visualisations for a given dataset producing rough thumbnails of layouts, where they experiment with spatial arrangements, try out different encodings (position, size, colour, shape etc.), all with nothing more than their hands, a pen and a sheet of paper, or a whiteboard (see Figure 28.1, Figure 28.2).

They annotate their sketches with notes about their intended audience, their narrative goals, the emotions they want to evoke, and then they categorise and filter their sketches. As the course progresses so to does the depth of engagement and interrogation of these ideas (see Figure 28.3, Figure 28.4).

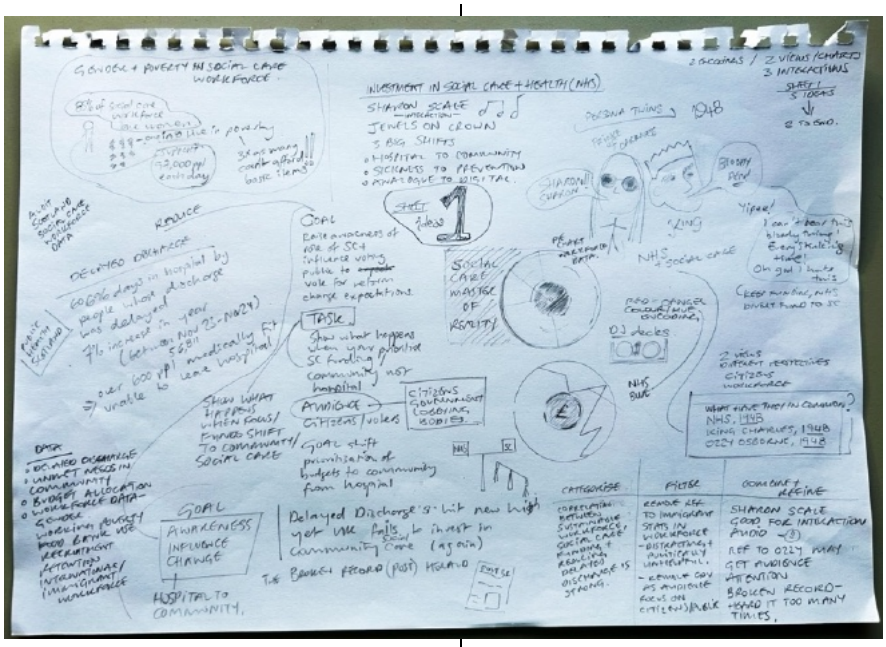


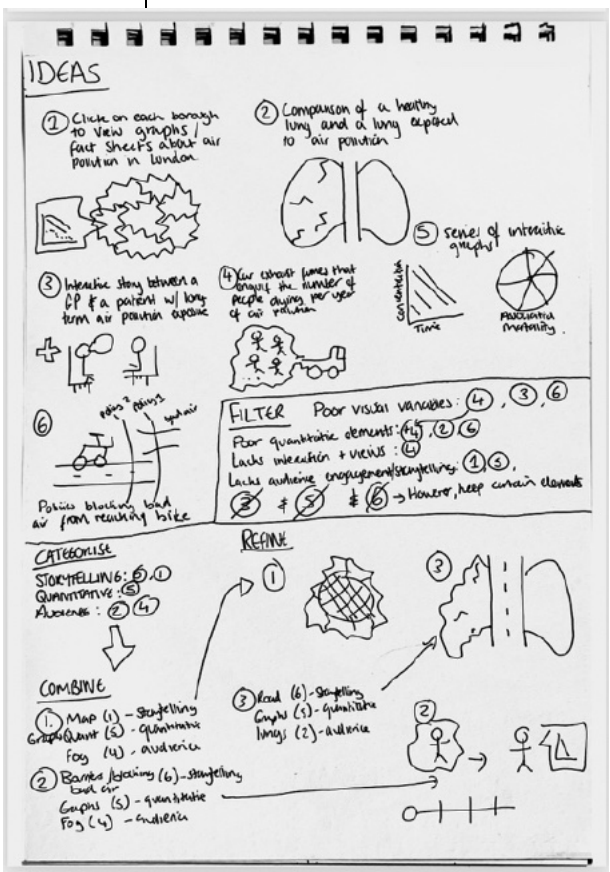
Figure 28.3: The rawness of a hand-drawn sketch is not a deficiency; it is a feature.

Supporting students through this process requires care, especially for those who feel their drawing skills are inadequate. For this reason, it is important to emphasise early and often that this is not about artistic quality – the doodles, the drawings, the sketches themselves are not being judged. The goal is ideation, not illustration. A visualisation sketch can consist of boxes, arrows, and rough axes. It can include figures and symbols, whatever the student wants, whatever their creativity allows; it does not need to be beautiful. One practical strategy that works well is the 'quick sketch challenge' where students are given a dataset, asked to identify an audience and a goal, and given approximately ten minutes to produce as many different visual representations as they can, without overthinking or self-censoring. The time pressure liberates them from perfectionism and opens a space for unexpected ideas.

Following this initial 'quick sketch', the sketches are critiqued through a modified 'gallery walk', where students are asked to share their sketches on the whiteboard, in the case of online students, or in the classroom, pinned to a wall, or spread out on desks. The students 'visit' each sketch, either leaving 'sticky-note' comments on each other's work or, if the group is small enough, sharing comments in real time. This activity transforms the individual exercise into a collaborative conversation, normalising the rawness of the work, and building a shared vocabulary for discussing what makes a visualisation effective or ineffective. Students begin to articulate, often for the first time, why a particular choice works, or even how elements from different options could come together to work even better.

This engagement, sharing, and articulation are where deep learning happens, and for many students 'pen and paper' turns out to be transformative in a way they had not expected. Students have stated that this step has changed the way they approach problems more broadly, and by the end of the course many report that they now sketch before they code, think through the design before worrying about the implementation, and seek to consider every component and aspect of the visualisations they create or view (see Figure 28.5, Figure 28.6, Figure 28.7). The pen, it happens, is not a step backwards but rather a step deeper.

Figure 28.4: The goal is ideation, not illustration.



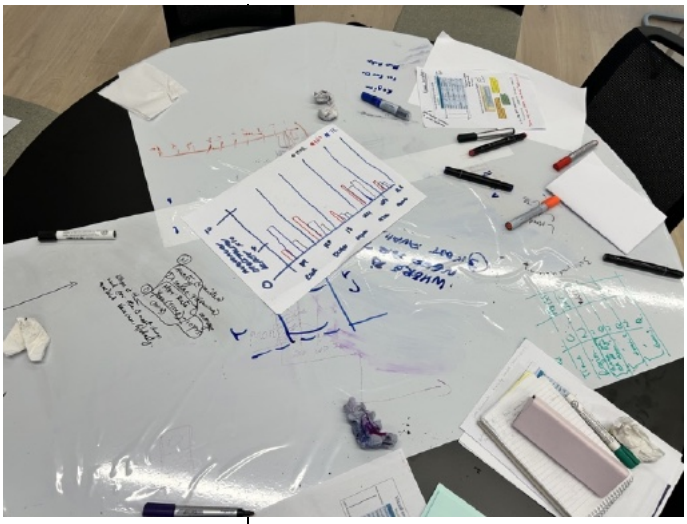


Figure 28.5: Normalising the rawness of the work.

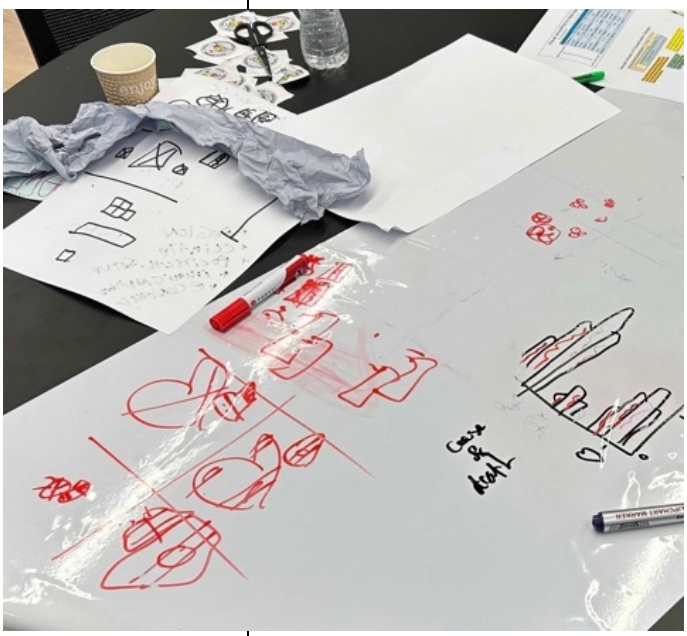


Figure 28.6: The time pressure liberates one from perfectionism...

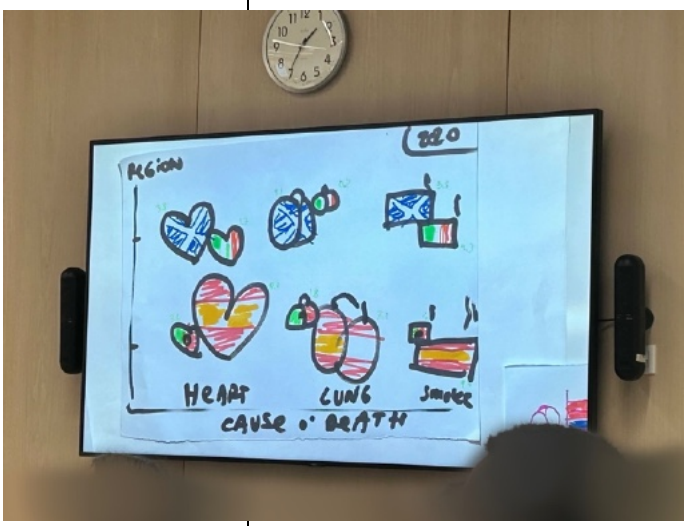


Figure 28.7: ...and opens a space for unexpected ideas.

Theory as a Foundation: Bertin, Tufte, and Munzner

The sketching, doodling and ideation work does not happen in a vacuum and is underpinned by a rigorous engagement with the theoretical foundations of data visualisation. These foundations, once understood, give students a language for making and evaluating their design decisions and the decisions of others. The course begins with Jacques Bertin, the French cartographer and theorist whose 1967 work *Sémiologie Graphique* remains one of the foundational texts of visualisation theory (Bertin 1983).

Bertin (1983) identifies, what he calls, visual variables, which are the fundamental perceptual properties used to encode information. These are position, size, shape, value (lightness), colour hue, orientation, and texture. His message is that not all variables are equal, with some better suited to encoding quantitative information (position and size are highly effective), while others work better for categorical distinctions (shape, colour hue). Understanding Bertin's visual variables means understanding the grammar of visual encoding, and once students have this grammar, they have the means to analyse any visualisation they encounter with a clarity and precision not previously possessed. While there have been many additions and variations made to Bertin's original work (and this is ongoing in this developing area of research), in this course the aim is to keep it as easy and workable for the students as possible with their focus on learning and using Bertin's visual variables as their foundation.

The course also introduces students to Edward Tufte, whose work is in many ways the 'aesthetic conscience' of data visualisation. Tufte's principles of data-ink ratio, 'chart-junk', small multiples, and the lie factor, are not merely stylistic prescriptions, but also ethical commitments (Tufte 1983). Tufte argues, compellingly, that a poorly designed visualisation is not just ugly; it is dishonest, it misleads readers, distorts understanding, and wastes attention. His concept of 'chart-junk' (the visual clutter that adds no informational value) resonates powerfully with students. Encountering Tufte can feel like a minor revelation where suddenly less is not just more in the aesthetic sense, it is also more from an ethical perspective. Of course, we do discuss the exceptions to these rules (see Lang (2009), for example) and challenge students to consider this and when it might be proper to break these rules.

The third major theoretical framework the course introduces is Tamara Munzner's nested model of visualisation design, first articulated in her 2009 paper and later expanded in her seminal textbook (Munzner 2009, 2014). Munzner's model is particularly valuable for students who are learning to think creatively and systematically about design decisions. It proposes four nested levels of the design process: characterising the domain problem and the data, abstracting those into visualisation tasks and data types, designing the visual encoding and interaction idioms, and finally implementing the algorithms and data structures (students do not realise this fourth level in this course). For students, it is not only what the model offers but how it functions that is important because the nested design model identifies that a mistake can occur at any level and that a mistake made at an outer level will cascade inward, invalidating all the work that follows. Munzner's model is a theoretical scaffolding that teaches students to question their assumptions at every stage, to ask not just how to create a visualisation but whether the visualisation they are creating is the right one for the task and for the audience. This is not a question that programming tools or AI tools ask. It is a question that only a thoughtful practitioner can ask if they have learned how to ask.

Woven throughout this visualisation theory is an engagement with perceptual psychology. Visualisation is a perceptual act where data is visually encoded and then decoded by a human's visual system, which has particular strengths and limitations. Drawing on foundational work in the field (Ware 2012), students learn about pre-attentive processing (the fact that certain visual properties such as colour, size, and orientation are detected almost instantaneously, in parallel, and before conscious attention is engaged), and the implications of this for design. They learn about the limits of human working memory, about the gestalt principles of visual grouping, about how colour is processed differently depending on whether it encodes categorical or quantitative information. This knowledge does not just make them better designers; it makes them more attentive ones. The human visual system is extraordinary, but it is also fallible and biased, and a visualisation that ignores its limitations is a visualisation that will fail its audience (Mandal 2003).

Storytelling and Audience: The Communicative Imperative

If theory gives students the vocabulary and doodling and sketching gives them the freedom, the course's emphasis on storytelling and audience gives them their purpose. A visualisation is always a communication. It is always made by someone, for someone, about something, in service of some goal. The moment students internalise this, not as an abstract idea but as a design constraint that shapes every decision, their work changes dramatically.

Audience is one of the key concepts that the course begins with on day one and returns to again and again. Different audiences bring different prior knowledge, different expectations, different emotional responses, and different purposes. In the specific context of health and social care data these distinctions carry a great deal of weight. A visualisation designed for a room of clinical specialists or public health researchers needs to do completely different work from one designed for patients, carers, or a general public audience, even if both are communicating the same underlying data. The specialists can handle complexity, density, and technical annotation; the general audience needs clarity, guided narrative, and context (see Figure 28.8, Figure 28.9). Getting this wrong is not merely a stylistic failure. It is a communicative failure which, in health contexts, can contribute to genuine misunderstanding and poor outcomes.

This course, therefore, does not treat audience as a single spectrum from novice to expert, as to do so would run the risk of missing out vital elements. Instead, students are taught to approach audience with curiosity. Not only who are the audience, but also what is the audience doing? Are they exploring data freely, forming their own questions and following their own paths? Or are they being presented with a specific argument, guided through a specific narrative? These two modes, one exploratory, the other explanatory, require fundamentally different design approaches. An exploratory visualisation needs interactivity, flexibility, and rich detail while an explanatory one needs clarity, focus, and narrative momentum. Many students arrive never having made this distinction before, and making it opens up an entirely new dimension of design thinking, which for many is transformative.

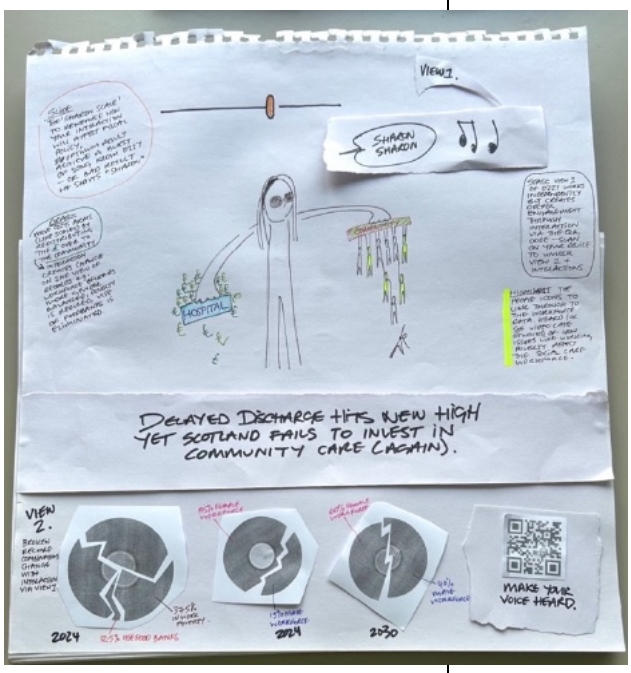


Figure 28.8: Theory gives the vocabulary; doodling gives freedom; storytelling gives purpose.

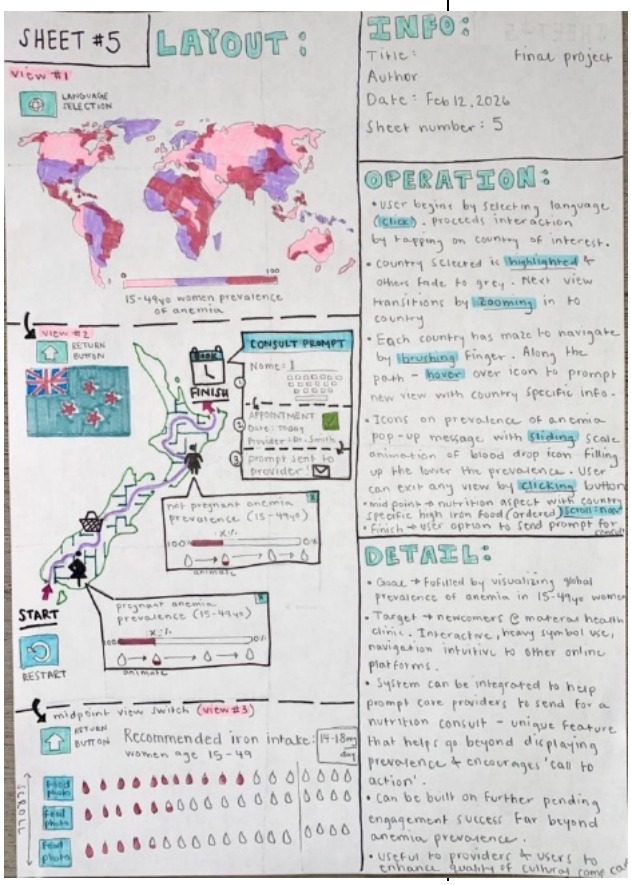


Figure 28.9: Experts can handle complexity, density, and technical annotation; the general public needs clarity, guided narrative, and context.

Storytelling, in this context, is not about imposing a narrative on data, rather it is about finding the truthful, meaningful shape of a dataset, the arc of change over time, the contrast between groups, the outlier that demands explanation, and presenting these things in a way that guides the audience towards understanding. The best data visualisations have a kind of rhetorical structure beginning somewhere, developing towards a point, and leaving the reader with something they did not have before. Students learn to ask, "what is the one thing I want my audience to take away from this?" before they begin their design doodling. This single question, explored and answered, transforms the entire process.

Practical activities support student learning and help them appreciate the importance and complexity of audience and goal. For example, students are asked to present visualisations they have designed to each other and to explain the narrative they intended, then to compare that intended narrative with what their audience actually took away. The gap between the two is almost always instructive. It reveals the biases we hold and the assumptions we make about what is obvious, what needs labelling, what requires annotation, what needs changing altogether. It teaches that what is clear to the maker is not always clear to the viewer and that this gap is the data visualiser's responsibility to close.

Critical Thinking as a Superpower

One of the most significant outcomes of the course, and one that students themselves often identify as surprising, is the development of critical thinking skills that extend far beyond data visualisation. The combination of theory, practice, audience awareness, and reflective practice that the course requires supports students to become not only more skilled visualisers, but also more thoughtful practitioners of data science in general.

Critical thinking, in this context, means the capacity to question the choices embedded in any artefact. To ask not just whether it works but whether it is the right thing to create, for whom it is being created, and what end it serves. Students learn that data visualisations are not neutral. Every design choice, what to include and exclude, how to scale axes, which colour palette to use, whether or not to add symbols, whether to annotate an outlier or let it speak for itself, whether interactions are required and what these might be, how they might be used and the impact they might have, is a choice that encodes assumptions about what matters and who will ultimately understand it. Students who have worked through Bertin's visual variables and Tufte's principles and Munzner's nested model, and have engaged with perceptual theory and the imperatives of understanding audience, are, by the end of the course well-equipped to ask questions with a rigour that students who have only learned to code are not.

The habit of critique, once established, is generative. Students who have learned to analyse and question visualisation design are in a position to transfer these skills to their study more broadly. They are more likely to question a technical output that seems analytically sound but communicatively misconceived. They are, in short, more conscious practitioners because they have learned to think, to see, to question … and to doodle. There is a developing creativity to be seen in students who have been through the process of doodling and sketching, of generating multiple solutions to a design problem before committing to any of them. These students begin to develop a fluency with divergent thinking that serves them across study and work domains. They are less likely to accept the first solution they think of, or are presented with. They are more comfortable with ambiguity and iteration, they demonstrate a higher tolerance for productive uncertainty, and they begin to see that creativity has a very central and important place in programming.

Future-Proofing in the Age of Generative AI

It would be easy, in 2026, to ask whether a course about data visualisation needs to teach design thinking when generative AI tools can now produce a working chart from a natural language prompt in seconds. Type a description into a capable AI system and you will have something render-ready almost instantaneously. If the code can write itself, why should students spend time learning to think about design? The answer, paradoxically, is that generative AI makes this kind of teaching more important, not less. The bottleneck in data visualisation has never been the ability to write code. It has always been the ability to think about what to create. A practitioner who can prompt an AI system to produce code is only as capable as their underlying design thinking. If they do not know that a bar chart is the wrong choice for this data and this audience, the AI will produce a bad bar chart very efficiently. If they do not know that the colour palette they have chosen is inaccessible to people with colour vision deficiencies, the AI will generate those inaccessible colours without hesitation. The code is the easy part, the thinking is the hard part, and it is the part that no AI can do for you.

So, in a world where AI can produce competent, adequate visualisations on demand, the value of truly excellent visualisation thinking increases dramatically. Competence is no longer scarce, but what is scarce, and will likely become scarcer, is the kind of deep understanding that allows a practitioner to produce visualisations that are not just technically correct but genuinely illuminating, genuinely communicative, and genuinely 'fit for purpose'.

There is also a need for critical literacy as AI-generated visualisations proliferate in public health communications, in policy documents, in clinical and even in research publications. The ability to evaluate and critique these visualisations becomes an even greater necessity. A

practitioner who has worked through Bertin and Tufte and Munzner, who has doodled and drawn, who has sketched dozens of alternative representations of a single dataset, who has presented their work to an audience and encountered the gap between intention and reception, who understands perceptual theory and how they apply, is well-placed to look at an AI-generated visualisation and ask the right questions. They will identify the misleading axis, the inappropriate chart type, the missing context, the distorted scale, the perceptual inadequacies, the missing audience.

Data Visualisation: Knowledge Transfer is a course that takes an active approach to future-proofing by positioning itself not to compete with AI tools but to develop the human capacities that AI cannot replicate. Students who complete this course are better positioned to work with AI-powered visualisation tools than those who have only ever learned to code – they know what to ask for, how to evaluate what they receive, and how to push beyond adequacy towards excellence.

Seeing Differently: The Lasting Transformation

The most powerful testimony to the value of this course comes from the students themselves, and it tends to take a particular form. They do not say that they learned new tools or mastered new techniques, though they do those things too. What they say, more often, is that they now see differently:

> "Prior to this course, my undergraduate background led me to believe that data visualisation was only concerned with selecting the correct type of graph for a given dataset. My understanding was limited to presenting data in its rawest form, without considering how it might be communicated to the public or how they may interpret it. I now realise the power and influence data visualisation holds in communicating public health issues." - Student 1

They look at a visualisation in a health report, a scientific paper, or a public dataset release, and they cannot help asking the questions they have been trained to ask. Whose audience is this for? What story is being told? What encoding choices have been made, and are they the right ones? What has been left out, and why?

> "I can no longer look at visualisations the same way. I find myself looking and thinking, hmmm that is not really the best encoding if you want the audience to act on this. This could be so much more effective if..." - Student 1

> "I now have an increased awareness of my own position when critiquing visualizations and understand that like in art, some of the most memorable pieces occur when the artists vision resonates deeply with viewers as the piece delivers some form of universal experience with the users. Though the individual experiences may be different, the messaging is clear, which is what makes it impactful." - Student 2

The course aims to produce not just students who can make good visualisations, but students who understand, at a deep and principled level, that visualisation is a form of knowledge transfer, a communicative act, a designed object that encodes both data and values. Students who understand this are better data scientists and better analysts because they understand the human dimension of everything they create:

> "I challenged myself to move away from my natural instincts and reliance on line graphs and bar charts and to apply my developing understanding of visualisation principles, such as storytelling and emotive encodings. I found this to be an intensive learning experience as it encouraged me to explore multiple combinations and ideas, which is something I have never done before when creating graphs." - Student 1

The experience of this course demonstrates that creativity is not a talent you either have or don't have; rather it makes clear that creativity is a skill that develops through practice, through the willingness to step outside of one's 'comfort zone', and the ability to open up one's mind and find value in the generation of imperfect ideas and learn from them. Students who arrive believing that they are not creative people (a belief that many data-focused students hold) often leave having revised that self-assessment entirely. They have discovered that creativity is not only or always about producing something beautiful from nothing, but rather it is about asking good questions, generating multiple possibilities, and making principled choices:

> "At times I found myself feeling stuck or insecure as I did not feel 'creative' enough to create something meaningful. However, this process forced me to think outside of how I normally do and push myself, which I really enjoyed." - Student 1

Throughout the course students learn to be conscious practitioners who are aware of what they are doing and why, who can articulate the principles behind their choices, and who can evaluate their own work and the work of others with genuine rigour:

> "[T]he goal of the final visualization was to inspire motivation within the users by being able to see themselves in the data, engage with the idea[s]... and advocate for themselves by increasing feelings of empowerment. In using affective visualization techniques, the final visualization optimizes users' engagement through exploring different sensations, narratives, and behaviours within the given context...The visualization allows messaging based on users' choices. They choose the setting, and the visualization encourages interaction through play. The activity enhances the storytelling as it symbolizes a journey from displaying ... information, to suggesting what the user can do

> with the information…" - Student 2

In the field of health and social care data, where visualisations shape how clinicians, policymakers, and the public understand outcomes, inequalities, and risk, that consciousness is not a luxury, it is a responsibility.

Conclusion: The Pen as Portal

There is something quietly radical about a course that begins by asking postgraduate students to put down their computers and pick up a pen, to draw and to doodle. It runs against the grain of a discipline that tends to value and promote technical sophistication, to measure progress in terms of tools mastered and programmes learned. *Data Visualisation: Knowledge Transfer* asks students to be uncertain, to be slow, to be raw and exploratory in an educational culture that often rewards precision and speed. Yet, the radicalism of this is inseparable from its value. By stepping back from the keyboard, students gain access to a creative and critical space that programming tools cannot fully offer. By engaging with the theories of Bertin (1983), Tufte (1983), and Munzner (2009, 2014), students acquire a vocabulary for thinking about visual communication. By attending to perceptual psychology (Ware 2012), audience, and narrative, they develop an understanding of visualisation as a human act, not merely a technical one. And by sketching, doodling, annotating, and critiquing, they build the habits of active, reflective practice that will serve them for the rest of their careers:

> "Initially, I viewed data visualisation as a neutral tool, focusing on clarity and function, but I now appreciate the power of emotionally driven storytelling. My final visualisation embodies this shift. Instead of conventional graphs, I used cultural iconography and metaphor, choosing Ozzy Osbourne to represent NHS and social care funding imbalances. Born in 1948, the same year the NHS and social care were established, Ozzy's chaotic yet beloved public image mirrors the precarious state of social care today…This course has given me the confidence to embrace a more human-centred and creative approach to data visualisation. I will use this in my work…" - Student 3

Data Visualisation: Knowledge Transfer is, at its core, a course about learning to see. It teaches students to see data, not as a collection of values to be rendered, but as a story waiting to be told. It teaches them to see their audience, not as a passive recipient of information, but as an active participant in a communicative act. And it teaches them to see themselves not only as technicians, but as designers making creative, responsible, and effective choices. In the age of generative AI, when the code can write itself, learning to see is more important than ever. The best tool for learning to see is not a programming environment or an AI assistant instead; it is a pen, a piece of paper, a whiteboard, and a willingness to think, to challenge … and to doodle.

Acknowledgements

I would like to thank and acknowledge the work and inspiration of my friend and colleague Dr Shane Sheehan whose enthusiasm, knowledge and undying curiosity and belief in the visual inspired and realised the development and growth of the course that this chapter is based on. I would also like to thank Dr Pawel Orzechowski for his support with this course in its most recent iteration. His enthusiasm and playful approach to teaching and learning has added yet another exciting dimension to this course. Also, a very big thank you to all who have participated in this course over the years. It is your enthusiasm together with your experiences that have helped shape this course and bring it to where it is today. Finally, a very heartfelt thanks to the students who so willingly and generously allowed me to include their work and reflections in this chapter. They bring the story of this course to life. Thank you!

References

Bertin, Jacques. 1983. *Semiology of Graphics*. University of Wisconsin press. https://doi.org/10.2307/143508.

Lang, Alexander. 2009. “Aesthetics in Information Visualization.” *Trends in Information Visualization* 8.

Mandal, Mrinal Kr. 2003. “The Human Visual System and Perception.” In *Multimedia Signals and Systems*, 33–56. Springer. https://doi.org/10.1007/978-1-4615-0265-4_3.

Munzner, Tamara. 2009. “A Nested Model for Visualization Design and Validation.” *IEEE Transactions on Visualization and Computer Graphics* 15 (6): 921–28. https://doi.org/10.1109/TVCG.2009.111.

———. 2014. *Visualization Analysis and Design*. A K Peters/CRC Press. https://doi.org/10.1145/3721241.3733989.

Tufte, Edward R. 1983. *The Visual Display of Quantitative Information*. 2nd ed. Cheshire, CT: Graphics Press. https://doi.org/10.1198/tech.2002.s78.

Ware, Colin. 2012. *Information Visualization: Perception for Design*. Morgan Kaufmann.

Part III — Assessment and Feedback: Principles and Practice

Umberto Noè

School of Philosophy, Psychology and Language Sciences, University of Edinburgh
Umberto.Noe@ed.ac.uk

Franziska McManus

Edinburgh Medical School, University of Edinburgh
z.mcmanus@ed.ac.uk

Brittany Blankinship

Edinburgh Medical School, University of Edinburgh
b.blankinship@ed.ac.uk

Assessment is, for many students, the most anxious part of their studies. For many educators, it is the most time-consuming. For institutions, it is one of the most consequential as assessment shapes what students prioritise, how they feel about their learning, and whether they can progress. This part of the book asks: what does good assessment and feedback look like in programming education, and how do we actually build it?

The chapters here approach these questions from two angles. The first theme looks at the *frameworks* we use to think about assessment and feedback, i.e. the underlying schools of thought and structures that shape what we measure and how we respond to student work. The second theme turns to how to do this *practically* by looking at how do educators design, deliver, and scale assessment in programming courses, particularly as generative AI and automated tools are changing the landscape rapidly.

Theme — Frameworks for Assessment and Feedback

Developing a Skills-Based Framework for Assessments (Pankratz 2026)

What if, instead of averaging marks across a course, students simply had to demonstrate each skill and could keep trying until they did? That is the core idea behind the skills-based assessment framework discussed in this chapter, and the author makes a compelling case for it in the context of teaching programming to novices. The author, a lecturer in Psychology at the University of Edinburgh teaching statistics using R, argues that traditional grading harms students who are already anxious, behind the pace, or facing structural disadvantages. The skills-based framework (where students earn skills permanently, are not penalised for re-attempts, and receive feedback on every attempt) shifts the dynamic entirely. The chapter includes a detailed account of how the author's team is redesigning their first-year statistics course around this framework.

This chapter is for you if you teach programming to students who did not come for the programming (e.g., psychologists, social scientists, engineers), and want a principled alternative to marks-based grading. It is also an interesting reading if you care about equity in your classroom as the chapter examines the disadvantages faced by many students and argues that the flexibility of skills-based assessment is not just to be considered pedagogically but also ethically.

The Action Feedback Protocol (MacLaren et al. 2026)

Feedback is only useful if students act on it. That sounds obvious, but many feedback systems are not designed with that goal in mind, they are often designed to justify marks. The Action Feedback Protocol (AFP), introduced in this chapter, is a deliberate attempt to

change that. Developed at Heriot-Watt University and now in use across disciplines and campuses worldwide, the AFP works on three levels: it trains students to receive feedback well ("Tune the Ear"), gives markers a simple and consistent structure for writing useful comments ("Simplify the Message"), and prompts students to do something with what they have read ("Encourage Action"). The chapter is written by the team who built the protocol, including a former student who was in one of the first cohorts to use it.

This chapter is for you if you have ever written detailed feedback that students ignored, or struggled to get a marking team to produce consistent comments. The AFP is refreshingly practical. It is not a theory of feedback but a working protocol, and the chapter shows how it has been implemented in a way that you can adapt to your own course.

Theme — Practical Methods for Assessing Programming

Assessment and Marking in an Introductory Programming Course for Physics Students (Waugh, Chislett, and Dash 2026)

What does a thoughtfully designed sequence of assessments actually look like, in full detail, for an introductory programming course? This chapter provides exactly that: an honest and detailed account of how assessment is implemented in a first-year computing course for physics undergraduates at UCL. The authors describe their full assessment framework (i.e., weekly programming tasks marked in person, online multiple-choice quizzes used as a pre-requisite for submission, take-home assignments, and a final computational physics project), and explain the reasoning behind each choice. They are honest about what has worked, what has not, and how the course has evolved year by year. The in-person marking model is particularly unique as it allows for immediate two-way feedback and, increasingly, offers a practical response to integrity concerns in an age of generative AI.

This chapter is for you if you are designing or revising an introductory programming course and want a detailed model to learn from. The authors do not claim to have found the perfect system. They share a system that has been tested, adapted, and improved. You will find practical insights on scaling feedback, managing marking workloads, and balancing formative and summative assessment in a course where programming is one component among several.

Generative AI in Assessments (Clift and Petrovska 2026)

Generative AI has changed what assessment design, marking, and feedback can look like, and this chapter is a practical guide to using it better. The authors walk through the full lifecycle of an assessment:

- using large language models (LLMs) to draft multiple-choice questions, write assignment briefs, build marking rubrics, and generate model solutions
- supporting students as an exploratory tool during the work itself
- and using AI to speed up the writing of personalised feedback at the end.

The authors' approach is LLM-agnostic and committed to human oversight, making it clear that the chapter sees GenAI as a tool for educators, not a replacement for their expertise.

This chapter is for you if you are curious about how AI can reduce the workload of assessment design and feedback without compromising quality or fairness. Even if you are sceptical about AI in teaching, the chapter's structured, step-by-step prompting examples make it easy to experiment. It is especially relevant for large courses where creating varied assessments and providing timely feedback at scale is a challenge.

Using Automated Marking in Programming Courses to Enhance Learning (Shafti and El Gemayel 2026)

Automated marking has been transforming programming education for years, and this chapter is a practical introduction to what it looks like in practice. Focusing on CodeRunner (a widely used Moodle plugin that runs student code against predefined test cases and returns instant feedback), the authors explain how to set up automated assessments, configure test cases, and customise the system for different languages and course contexts. The chapter draws on experience from the University of Strathclyde, where automated marking has been used across Computer Science and non-Computer Science courses at multiple undergraduate levels, and acknowledges both the benefits and the learning curve involved.

This chapter is for you if you are considering automated marking for the first time and want to hear the reality of implementing it, rather than a sales pitch. It is also useful if you already use automated tools and want to extend what you do, as the chapter covers advanced configurations and points to further resources. If your students are not computer scientists and have little prior experience of programming, the setting described by the authors will feel particularly familiar.

Happy reading!

References

Clift, Lee, and Olga Petrovska. 2026. “Generative AI in Assessments: Before, During, and After.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

MacLaren, Andrew, Frederick Madsen, Antonia Voigt, Alex Buckley, and Tom Farrington. 2026. “Managing the Rubik’s Cube of Assessment: The Action Feedback Protocol.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Pankratz, Elizabeth. 2026. “Developing a Skills-Based Framework for Assessments in Programming Courses.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Shafti, Leila S., and Joseph El Gemayel. 2026. “Using Automated Marking in Programming Courses to Enhance Learning.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Waugh, Ben, Rebecca Chislett, and Louise Dash. 2026. “Assessment and Marking in an Introductory Programming Course for Physics Students.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Developing a Skills-Based Framework for Assessments in Programming Courses

Elizabeth Pankratz

School of Philosophy, Psychology, and Language Sciences, University of Edinburgh
elizabeth.pankratz@ed.ac.uk

In This Chapter

To learn to program is to develop a new set of skills. In this chapter, I'll discuss an approach to course design and assessment that brings skill development to the fore: the skills-based assessment framework. I'll explore what makes this framework particularly good for teaching programming to new programmers, especially ones facing cultural, structural, and institutional disadvantages. And I'll share a case study from my own context: I work as a Lecturer in Psychology at the University of Edinburgh, teaching undergraduate Psychology students how to do statistics using R. With my colleagues, I have developed an introductory course for these students using the skills-based framework, and our experience will be the focus of the final part of this chapter.

What Is Skills-Based Assessment?

A course built for skills-based assessment comes with an inventory of skills that the course developer wants their students to learn. For example, in our intro statistics course, skills include "I can represent binary predictors using treatment coding" and "I can transform a variable into z-scores"; in a more coding-focused course, skills might be something like "I can explain the difference between while loops and for loops" or "I can use pseudocode to write basic sorting algorithms".

The basic idea behind one possible variant of skills-based assessment, the one my team and I are using, goes as follows. As the course progresses, students work on developing each skill. They get multiple opportunities to show that they've gained each one. The only firm deadline is the very end of the course; until then, students can try to demonstrate a skill as many times as they need to or want to, and they won't be penalised at all for making multiple attempts. They get feedback on every attempt—why it succeeded or where it fell short. Once a student has successfully demonstrated a skill, then in the course's record-keeping, they have earned that skill permanently and cannot un-earn it. Each student's course outcome (whether number grade, letter grade, pass/fail, or something else) is some function of how many skills they earned.

This basic idea is known in different circles under different names. For example, Clark and Talbert (2023) and Buckmiller, Peters, and Kruse (2017) call it "standards-based grading"; O'Leary and Stockwell (2021), (2022), and Zuraw et al. (2019) call it "skills-based grading"; and it's also similar to a cousin framework called "specifications-based grading" (Nilson 2014; Clark and Talbert 2023).

I like the "skills-based" part of the name because it centres the skills, but I steer away from the "grading" part for a couple reasons. First, the skills-based course that my colleagues and I are developing will culminate in a pass/fail outcome, not a number/letter grade. I don't want the term "skills-based grading" to mislead students about how they'll be assessed. Second, I consider grades on the whole to be an unnecessary evil (see, e.g., Kohn 1993; Stommel 2023; Blum 2020), but "assessment" can be about assessing how a student is doing, about checking in and giving feedback on their learning journey—an approach which better encourages genuine learning (see, e.g., Ambrose et al. 2010; Sambell, McDowell, and Montgomery 2012).

Why Is the Skills-Based Framework Well-Suited for Teaching New Programmers?

In this section, I'll highlight three features of skills-based assessment that, in my view, make it much better than traditional weighted-averages grading for teaching programming to new programmers. Skills-based assessment:

1. reduces students' academic anxiety and stress;
2. lets students learn at their own pace; and
3. helps students facing structural disadvantages to succeed.

1. Reduces Students' Academic Anxiety and Stress

Many students, especially those from non-STEM disciplines, are anxious about learning to program (an observation to which a whole chapter in this volume is dedicated: Oldnall et al. 2026). And learning to program in a traditionally-graded course is like adding salt to the wound. Learning is messy, and mistakes are normal—arguably essential—along the way. But in traditionally-graded courses, mistakes are punished by lowering marks.

On the other hand, the skills-based framework acknowledges and encourages the messy process of learning. For example, allowing students the chance to demonstrate a skill repeatedly, with no penalty for re-attempts, fosters in them a sense that it's okay to make a mistake and worthwhile to try again (O'Leary and Stockwell 2021). And students also gain a sense of self-efficacy as they realise they can accomplish something that they couldn't do before (Zuraw et al. 2019). This leniency is a major reason why skills-based assessment has repeatedly been shown to lessen students' academic stress (e.g., O'Leary and Stockwell 2021; Buckmiller, Peters, and Kruse 2017; Zuraw et al. 2019; Lewis 2022).

In an ongoing research study, my colleagues and I are gathering data about how our skills-based course redesign impacts how our students feel about learning statistics and R programming. Based on the literature mentioned here, we hypothesise (and very much hope!) that our redesign will help ease our students' anxiety and stress.

2. Lets Students Learn at Their Own Pace

Programming can be hard to learn. And while some people will take to it straight away, others need more time to develop this new way of thinking (Caspersen and Bennedsen 2007; Offutt et al. 2017). But students in traditionally-graded courses don't always have the time they need. Students typically lose marks if they haven't grasped a concept by the time the course developer thinks they should have—even if they do figure it out later.

In skills-based courses, learners are not held to the course developer's timeline. Students can demonstrate skills at any point during the course, so learners who benefit from a bit more time to learn can still succeed. Offutt et al. (2017) report that letting students pace their own learning in an introductory computer science course has led to great results: students learn better and are reportedly disinclined toward academic dishonesty, because the time pressure that had motivated them to cheat has been lifted.

3. Helps Students Facing Structural Disadvantages To Succeed

The culturally ideal programmer is an able-bodied white man. Anyone who diverges from this ideal, along whichever dimension or dimensions, has extra cultural and emotional work to do when learning to code.

For example, Carter and Jenkins (1999) observed a "growing perception that female students are weaker at programming than their male counterparts" (p. 3), noting that female students tend to be less confident than male students and to underestimate their own ability more. In my earlier experience as a young female programmer, and now as a female teacher with many young female students, not much has changed about these views in the nearly three decades since.

And adding in the intersectional experience of race, Rea (2022) recounts the striking words of Olivia, who organises coding bootcamps for marginalised and minoritised people:

> I'm teaching women how to code. But it's bigger than that. What I'm really teaching them to do is how to unlearn what they've been taught. A lot of women of color, or women in general, have not been taught that science, math, coding, you name it, is for them. You're unlearning a thing, and you're learning a new skill.

Further dimensions compound the challenge of belonging in traditional academic spaces, no matter what's being taught: physical disability, mental disability, care responsibilities, work duties, a different first language than the language of education. All of these disadvantages can present a serious barrier in traditionally graded courses, in which students are commonly graded on things that go beyond the course's content-related learning objectives, such as regular attendance, active verbal in-class participation, and (as mentioned above) the timeliness of their understanding.

But in the skills-based framework, barriers like these are softened. Where clear expectations and goals are provided, less background knowledge is required, making the space more inclusive to people with different backgrounds (Parker 2018).[132] As such, skills-based courses have been shown to achieve more equitable outcomes for a wider range of learners (e.g., by O'Leary and Stockwell 2021, 2022). I really hope that the more flexible structure of skills-based assessment will help our intersectionally disadvantaged students build their skills as well as their sense of self-efficacy in a more compassionate space.

Finally, a note for even the culturally ideal student of programming: any person at any time could experience sickness, injury, or grief, or may just need time away to, say, bring their cat to the vet. Everybody benefits from inclusive pedagogy.

For further discussion of inclusion and accessibility in teaching programming, see Colquhoun et al. (2026) in this volume.

Case Study: Developing a Skills-Based Course Introducing Statistics Using R

For all the reasons discussed above and more, my team and I in Psychology at the University of Edinburgh are using the skills-based framework to redesign our undergraduate statistics curriculum. The redesigned first-year course (link to the University of Edinburgh course catalogue DRPS)[133] will launch in September 2026, so as of this writing, no students have experienced it yet. In this section, I'll describe the path we took (largely guided by Clark and Talbert 2023, chap. 11) and some key decisions we made along the way.

Our Development Process

We began by dreaming up, in very broad strokes, what topics and methods we wanted to cover in our ideal version of the course. Our early notes included, for example, "intro tidyverse", "descriptive stats", and "simple linear model". Once we all agreed on a set of topics and a rough order in which they'd appear, we dove into the weeds to develop a finer-grained list of things we want students to be able to do, such as:

- Install packages in R.
- Create new columns in a dataframe.
- Summarise numeric variables using the mean.

We ended up with a list of skills numbering between 80 and 100. This list grew and shrunk and changed over time as our ideas kept flowing. These are far from the set of skills we'll actually assess, but this early representative list of skills was absolutely good enough for the steps that would follow.

At the same time, we also considered the course's submission and feedback systems. How and when can students evidence skills? What kind of feedback will they receive?

We observed that many skills could be automatically checked (for example, identifying the code that would accomplish a given task), while a handful would require human review (for example, writing up a description of a given data set).

We decided to check most skills automatically using quizzes on our learning management system Blackboard Learn. These quizzes pull from a pre-built question pool for each skill. If students answer a question correctly, then that counts as evidencing the given skill. After submitting their answer, students will see automated feedback about how to solve the question and common misconceptions.

We decided to check the remaining skills that required human review at two points, one at the end of each semester. Students would be able to submit their responses (also on Learn) at any time, but to keep marking overhead reasonable, we only review their responses and give feedback twice a year. The feedback will be uniform across all students and simple, something like "successful", "almost", or "not yet" (Clark and Talbert 2023). For more specific guidance, students will be able to talk to us during the weekly coding workshops and our office hours.

Once these basic structures were in place, we began properly consulting with stakeholders (Sackstein 2015). From the beginning, we had support from the head of our subject area, which made every other consultation much simpler. We asked for input from the following groups of people:

- Students (but don't expect a huge turn-out—we hosted an in-person drop-in feedback session, to which two of our approximately 400 students came, and we also sent out an online feedback form, to which four students responded).

[132] A well-formulated rubric may accomplish a similar goal, but rubrics tend to have two issues that skills do not. For one, by their nature, rubrics focus on the work's deficits until the very highest levels of achievement. Here is an example from one of our earlier report-based assessments: "Results are mostly clear and logically presented. Interpretation is mostly accurate but may lack depth in linkage to analysis strategy". A rubric targets all the ways the student's product falls short from the ideal. And for a student who already feels like they don't belong, deficit-focused approaches like this will not make them feel more welcome. And for another, imagine being a student and receiving the feedback: "to get a higher mark, you need to add more depth in the linkage to analysis strategy". What does it *mean* to "add depth"? And how will you know if you've successfully done it? Rubrics tend not to be as explicitly actionable as skills, and they tend to require background academic knowledge to successfully interpret—knowledge which educators take for granted but which not all students will have.

[133] https://www.drps.ed.ac.uk/26-27/dpt/cxpsyl08013.htm

- Academic staff (we hosted a staff-focused drop-in session and sent out an online feedback form, which altogether yielded a handful more responses from staff than from students).
- Professional services staff (especially the very knowledgeable people who deal with the back-end of inputting student grades into the institution's software system; they helped us identify and solve problems we had never considered).

After integrating the very useful input we got from all these people (examples of which I'll mention below), we submitted the proposal to the committee that processes course changes.

At that point, with the administrative overhead off our desks for a while, we started to fine-tune the skills that the first iteration of the course will actually assess.

Top Tips For Writing Assessment Skills

Designing the skills themselves is a creative logistical challenge with many degrees of freedom. In this section, I'll walk through how we resolved two major questions that came up repeatedly along the way: how specific vs. how high-level should each skill be? And what level of cognitive engagement should each skill call for?

Granularity of Assessment Skills

In our early list of topics, we wrote, for example, that we want students to learn to summarise data using the mean, the median, and the mode. How many skills should we use to assess this learning?

One option is to use one skill per measure:

- I can summarise numeric variables using the mean.
- I can summarise numeric variables using the median.
- I can summarise categorical variables using the mode.

Another option is to consolidate both numeric summary measures into a single skill:

- I can summarise numeric variables using appropriate measure(s) of central tendency.
- I can summarise categorical variables using appropriate measure(s) of central tendency.

A third option is to consolidate again:

- I can summarise variables using appropriate measure(s) of central tendency.

In principle, one could consolidate arbitrarily far, depending on the focus of the course: from "I can conduct an exploratory data analysis" up to "I can analyse data", or even beyond. Here, for the purposes of illustration, I will focus on the lowest three levels, since they are the most pertinent ones for our specific aims.

The relationship between these different levels of skills is shown in Figure 30.1, along with some useful vocabulary offered by Clark and Talbert (2023, 159).

At the very top of this hierarchy, in darkest blue, are what Clark and Talbert call the "course-level objectives": the broad, encompassing learning objectives for the entire course. Our running example belongs to a course-level objective like "I can summarise data", an objective which involves measures not just of central tendency but also of dispersion, for example.

Jumping to the very bottom of this hierarchy, in lightest blue, we see the "lesson-level objectives": what we want students to learn from individual lessons. Lesson-level objectives map fairly cleanly to the fine-grained list of learning objectives that we drafted early on.

In between these two extremes, in medium blues, we see the "assessment-level objectives". Somewhere in this middle ground are the skills that we will assess.

Do we want to make absolutely sure that students can summarise numeric and categorical variables individually? If we do, then a larger quantity of lower-

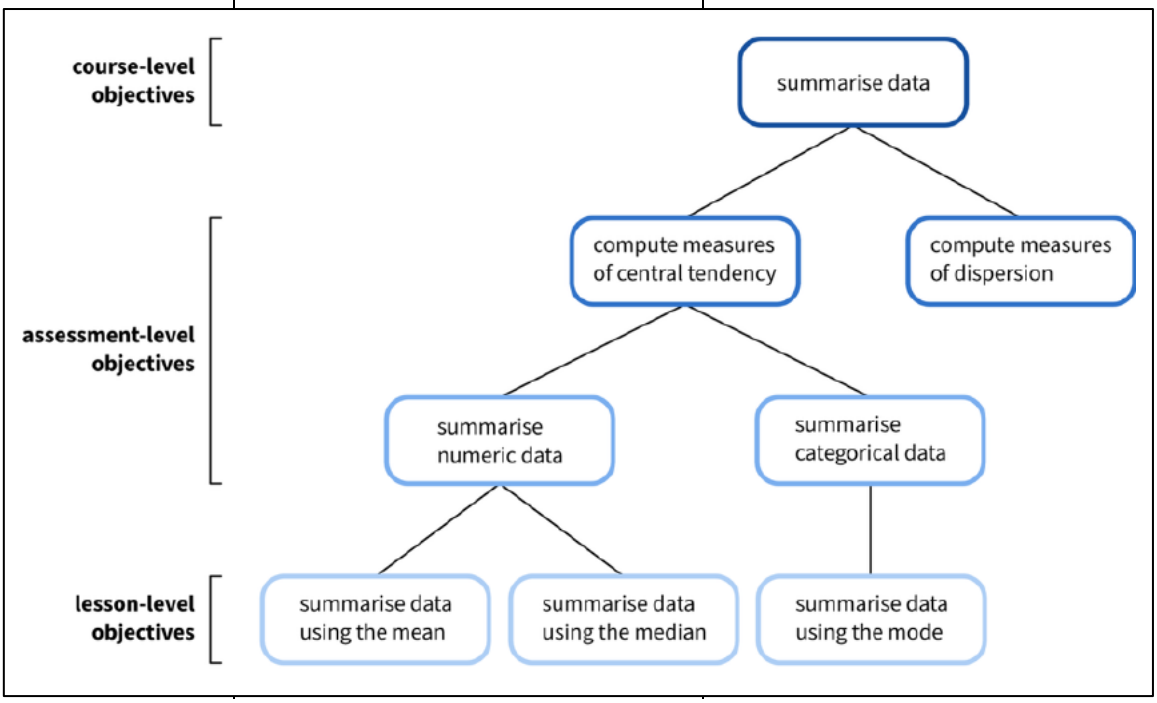


Figure 30.1: An example hierarchy of data summary skills, annotated with the levels described by Clark and Talbert (2023): course-level objectives, assessment-level objectives (a subset of which are the skills we want to assess), and lesson-level objectives.

level skills may be the way to go. Or do we assume that if a student can correctly find a sample's mean, say, they can probably also find the median and mode? If that assumption seems reasonable, then the higher-level skill might suffice.

In a coding course, an analogue to this quandary might be the following: Will students be assessed on appropriate use of control structures, broadly speaking? Or separately on being able to appropriately use if/then/else statements and various kinds of loops? Or, finer-grained, on specific structures like if/then, if/else, for loops, while loops, and so on?

In our case, for the data summary example, it is reasonable enough to assume that a student who can find the mean can probably also find the median and the mode. And further, we are also trying to keep the sheer number of skills in check (the current list, as of this writing, contains about 40). We therefore chose to assess the most consolidated, broadest, highest-level version of skills wherever we could.

Cognitive Level of Assessment Skills

Each skill should call for an action that a student can easily understand and that an assessor can easily observe (Clark and Talbert 2023). But for a given skill, we found that often, several different actions will all seem reasonable. For example, after students learn about t-tests, do we want them to be able to *conduct* a t-test, or to *explain* the logic behind a t-test?

Choosing the action verb is a weighty decision. Different verbs, like "conduct" vs. "explain", invoke different levels of cognitive engagement, as we know from frameworks like Bloom's taxonomy (Bloom et al. 1956; Anderson and Krathwohl 2001) and the SOLO (Structure of the Observed Learning Outcome) taxonomy (Biggs and Collis 1982).

In an early version of the course design, we had imagined that students could achieve one of three possible course outcomes: Merit, Pass, or Fail. This design allowed us two sets of skills: a "core" set of skills, which would cover simply conducting statistical tests, as well as an "advanced" set of skills, which would add on the deeper conceptual understanding. If students demonstrated both core and advanced skills, they would receive a "Merit" outcome; if they demonstrated only core skills, they would receive a "Pass". (I note that pedagogically, dangling the "Merit" carrot may have many of the same demotivating effects as traditional grading—see, e.g., Kohn 1993.) We discovered, though, that a categorical three-level course outcome is impossible within the strictures of our institution's software system—one example of many to illustrate why consulting with professional services colleagues is absolutely key.

In the later version of the course, we narrowed the outcomes down to Pass and Fail (an option welcomed by several students we talked to, who liked that it frees them from the stress of "chasing an A"). With no "Merit" category, we set aside the core vs. advanced sets of skills. Again, we faced a choice between "conduct" and "explain"—but again, Clark and Talbert (2023) offer guidance. They comment (p. 165) that hands-on, skills-building courses tend to use vocabulary from the more fundamental, less abstract levels of Bloom's taxonomy, whereas more conceptually-focused courses tend to use the more abstract, higher-level verbs. We want our students to focus on building skills, so we opted for the more hands-on "conduct a t-test" variant throughout.

The Big Skills-Based Picture

To conclude, let's zoom back out: what makes skills-based assessment well-suited for helping students learn to program? The picture I hope to have painted in this chapter shows a different way of checking in on how people learn, one that genuinely recognises students as the learners they are, not just as faulty repositories for the information we bestow (Freire 1996). Because skills-based assessment mitigates common causes of anxiety and softens many structural barriers that students face, it is our framework of choice for helping new programmers start their learning journey.

Acknowledgements

This chapter would not exist without:

- Franziska McManus, who first suggested I share these ideas in this book.
- Itamar Kastner, who introduced me to skills-based assessment and who continues to inspire the best parts of my practice.
- Emma Waterston, Josiah King, and Umberto Noè, my colleagues on the Psychology stats team, who are all incredibly caring educators and fun, thoughtful, and thought-provoking collaborators.

References

Ambrose, Susan A, Michael B Bridges, Michele DiPietro, Marsha C Lovett, and Marie K Norman. 2010. *How Learning Works*. San Francisco: Jossey-Bass.

Anderson, L. W., and D. R. Krathwohl. 2001. *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. New York: Longman.

Biggs, John B., and Kevin F. Collis. 1982. *Evaluating the Quality of Learning: The SOLO Taxonomy (Structure of the Observed Learning Outcome)*. Academic Press, Inc.

Bloom, Benjamin, Max D. Engelhart, Edward J. Furst, Walker H. Hill, and David R. Krathwohl. 1956. *Taxonomy of Educational Objectives: The Classification of Educational Goals*. Handbook 1: Cognitive Domain. London: Longman.

Blum, Susan D., ed. 2020. *Ungrading: Why Rating Students Undermines Learning (and What to Do Instead)*. West Virginia University Press.

Buckmiller, Tom, Randal Peters, and Jerrid Kruse. 2017. "Questioning Points and Percentages: Standards-Based Grading (SBG) in Higher Education." *College Teaching* 65 (4): 151–57. https://doi.org/10.1080/87567555.2017.1302919.

Carter, Janet, and Tony Jenkins. 1999. "Gender and Programming: What's Going On?" In *Proceedings of the 4th Annual SIGCSE/SIGCUE ITiCSE Conference on Innovation and Technology in Computer Science Education*. https://doi.org/10.1145/305786.305824.

Caspersen, Michael E., and Jens Bennedsen. 2007. "Instructional Design of a Programming Course: A Learning Theoretic Approach." In *Proceedings of the Third International Workshop on Computing Education Research*, 111–22. Atlanta Georgia USA: ACM. https://doi.org/10.1145/1288580.1288595.

Clark, David, and Robert Talbert. 2023. *Grading for Growth: A Guide to Alternative Grading Practices That Promote Authentic Student Learning and Student Engagement in Higher Education*. Routledge.

Colquhoun, Rebecca, Samantha Ahern, Brittany Blankinship, and Patricia Loto. 2026. "Practices to Foster Inclusion and Accessibility in Programming Teaching." In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Freire, Paulo. 1996. *Pedagogy of the Oppressed*. Translated by Myra Bergman Ramos. Penguin.

Kohn, Alfie. 1993. *Punished by Rewards: The Trouble with Gold Stars, Incentive Plans, A's, Praise, and Other Bribes*. Boston: Houghton, Mifflin and Company.

Lewis, Drew. 2022. "Impacts of Standards-Based Grading on Students' Mindset and Test Anxiety." *The Journal of Scholarship of Teaching and Learning* 22 (2). https://doi.org/10.14434/josotl.v22i2.31308.

Nilson, Linda B. 2014. *Specifications Grading: Restoring Rigor, Motivating Students, and Saving Faculty Time*. Sterling, VA: Stylus.

O'Leary, Maura, and Richard Stockwell. 2021. "Skills-Based Grading: A Novel Approach to Teaching Formal Semantics." *Proceedings of the Linguistic Society of America* 6 (1): 869. https://doi.org/10.3765/PLSA.V6I1.5025.

———. 2022. "Implementing Skills-Based Grading in a Linguistics Course." *American Speech* 97 (2): 247–62. https://doi.org/10.1215/00031283-9940629.

Offutt, Jeff, P. Ammann, K. Dobolyi, Christian Kauffmann, Jaime Lester, Upsorn Praphamontripong, H. Rangwala, Sanjeev Setia, Pearl Y. Wang, and Lisa N. White. 2017. "A Novel Self-Paced Model for Teaching Programming." In *ACM Conference on Learning Scale*, 177–80. ACM. https://doi.org/10.1145/3051457.3053978.

Oldnall, Chris, William Kay, Chris Sutherland, Tiago A. Marques, and Robert Young. 2026. "Overcoming Coding Anxiety: Lowering the Stakes and Making It Fun." In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Parker, Priya. 2018. *The Art of Gathering*. Penguin.

Rea, Ashley. 2022. "Coding Equity: Social Justice and Computer Programming Literacy Education." *IEEE Transactions on Professional Communication* 65 (1): 87–103. https://doi.org/10.1109/tpc.2022.3143965.

Sackstein, Starr. 2015. *Hacking Assessment: 10 Ways to Go Gradeless in a Traditional Grades School*. Vol. 3. Hack Learning. Times 10 Publications.

Sambell, Kay, Liz McDowell, and Catherine Montgomery. 2012. *Assessment for Learning in Higher Education*. Routledge.

Stommel, Jesse. 2023. *Undoing the Grade: Why We Grade, and How to Stop*. Hybrid Pedagogy Inc.

Zuraw, Kie, Ann M. Aly, Isabelle Lin, and Adam J. Royer. 2019. "Gotta Catch 'Em All: Skills Grading in Undergraduate Linguistics." *Language* 95 (4): e406–27. https://doi.org/10.1353/lan.2019.0081.

Managing the Rubik's Cube of Assessment: The Action Feedback Protocol

Andrew MacLaren

School of Social Sciences, Heriot-Watt University
a.maclaren@hw.ac.uk

Frederik Madsen

School of GeoSciences, University of Edinburgh; Lyell Centre, British Geological Survey
f.d.madsen@sms.ed.ac.uk

Antonia Voigt

Surrey Business School, University of Surrey
a.voigt@surrey.ac.uk

Alex Buckley

Learning and Teaching Academy, Heriot-Watt University
Alex.Buckley@hw.ac.uk

Tom Farrington

Storm ID
t.farrington@yahoo.com

A failing system or failure to recognise the system?

A university's approach to feedback reflects multiple aspects of its relationship with students. Decisions about assessment policy affect the whole institutional landscape, including retention, student experience, National Student Survey results, transparency, staff workload and student employment, as well as students' academic progress. Feedback is embedded within a system, and innovating in feedback benefits from acknowledging its multidimensional, interdependent features. However, conventional approaches to feedback tend to be limited because they are not framed within the broader institutional system. In this chapter, we introduce the Action Feedback Protocol (AFP), a systems-oriented approach to feedback that aligns student feedback literacy with a simple, efficient approach to producing feedback, all of which is directed towards encouraging the learner to take action.

We, the authors of this chapter have been involved in the development of the AFP since it began in 2019. We represent the tricky and interdependent moving parts of the feedback Rubik's cube, including: the students (Antonia was among the first student cohorts to use the AFP and has since completed her PhD in Education); the educators (Andrew and Tom are both Social Scientists, Frederik is a Geophysicist); and the institution (Alex is the assessment and feedback lead for Heriot-Watt's Learning and Teaching Academy). And a bit like a Rubik's cube, the AFP is also a 3x3 system: three dimensions that centre around three comments in written feedback.

Feedback is often considered an isolated event attached to the distribution of marks. Higgins, Hartley, and Skelton (2001), among many other researchers, find that this event-type posture stops feedback from being used to its full potential. On the learner side, Winstone, Nash, Parker, et al. (2017) and Winstone, Nash, Rowntree, et al. (2017) highlight the under-emphasised area of learner responsibility in the feedback dynamic, something they describe as a lack of 'proactive recipience'. The role of the learner is equally important in the ultimate effectiveness of the feedback they receive. Orsmond, Merry, and Reiling (2005, 381) note that if feedback "is simply stored in memory and never used, it is not feedback". This is particularly relevant in Science, Technology, Engineering and Mathematics (STEM) subjects, where reflective practice is not a part of the typical curriculum (Boswell 2023). Bevan et al. (2008) and Weaver (2006) attribute these issues to the common absence or under-provision of training for learners in the role of feedback in their learning. These conditions create the prevailing sense that more could be done to ensure that learners are adequately prepared to meet the feedback they receive halfway.



Figure 31.1: The Rubik's Cube of Assessment

Introducing the Action Feedback Protocol

Addressing this complexity therefore requires simultaneous attention on how students are informed on feedback literacy and how they are supported to act on the feedback they receive, as well as carefully considering how we, as educators, produce feedback in the first place. The AFP uses evidence-informed, simple and robust approaches to address feedback as part of a bigger complex system, rather than treating it as an isolated event. The approach therefore tackles head-on the need to foster proactive recipience among learners, couples the nature of feedback produced with students' expectations of it, and creates structured alignment between staff across marking teams and student programmes. The AFP's three dimensions are: (1) Tune the Ear, (2) Simplify the Message, and (3) Encourage Action.

Tune the Ear

This dimension involves focusing on improving learners' feedback literacy, through specific resources that help them understand the purpose and value of feedback. The resources include a professionally recorded podcast, a video, plus documents based on the Developing Engagement with Feedback Toolkit (Winstone, Mathlin, and Nash 2019).

Simplify the Message

At the centre of the AFP system is the 'three comments approach'. This involves providing feedback comments in a way that is concise, targeted and actionable by following a simple, evidence-based and efficient way of producing feedback comments. The comments are clearly signposted, are presented in a defined order, and use content that is Specific, Substantive, and Supportive (what we call the 'three Ss').

- **Comment 1 Motivates:** it identifies the main area (or areas) done well in the work and encourages the student to keep up that good practice in the future.
- **Comment 2 Informs:** it identifies the main area (or areas) not done so well that were specific to the assessment and explain how they could have been better. In written work, it may be helpful to quote an example from the submission and explain how it could be improved; in calculation-based work, suggesting an alternative formula or a more efficient approach to the calculations may be helpful.
- **Comment 3 'Feeds forward':** it identifies a point of process that is generally applicable to assessed work on the programme (or later in a specific course), suggesting ways in which that could be improved for future work. For example, if several errors relate to a foundational principle or law in the subject, directing a student to revisit that will help them in future work. Or, it may be a specific technique that is applicable to multiple assessment types, such as referencing or notation.

Encourage Action

This involves signposting students to help them reflect on the feedback, and encouraging them to take meaningful action on it as part of learning and improving (e.g. giving them access to resources such as a feedback portfolio and development tracker).

A Genuine Example of the Three Comments From Engineering

Aspect(s) done particularly well: A feature of this work that was done well was the presentation of a very detailed report in terms of the experimental work and its results. It demonstrates a good understanding of thermal systems and subsystem integration. The relevance to sustainable energy practices in Malaysia is well contextualised.

- **Aspect(s) that could have been done better:** An area in this work where there was room for further development was to expand on discrepancies between theoretical and experimental results. For example, the mismatch in sand heating times could be analysed in more depth, with specific proposals for reducing air gaps or improving pipeline design. Greater attention on the scalability of the system for real-world applications and how the economic feasibility would be addressed would be valuable things to include in the discussion.

- **A point of process that should be worked on for the future:** Something that might help you in future work would be to consider report writing technique, this is very important for professional engineers. Part of the report's job is to help the reader make sense of the meaning of results, not just report them. Working on this will improve your professional reporting practice. By continuing to improve your presentation (e.g. refer to all images and graphs from within the text, number equations, use a numbered (IEEE) system of standard referencing fully) you will develop skills that will stay with you throughout your engineering career.

- **Now Make an Action Plan For Your Feedback!** Now that you have read your feedback, it is important that you take action on it.

Feedback not acted upon is not really feedback, so now it is up to you to make an action plan to ensure you make the most of this feedback and use it in your future work. You can make your action plan by simply hitting 'add comment' below, or even better you can **download the feedback portfolio** on the 'Making the Most of Feedback' site[134]. This will help you make an action plan and track your progress across all your assessed work.

134 https://sites.google.com/view/makingthemostoffeedback/home

The AFP is designed to be as simple as possible for everyone: parsimonious and efficient for educators and, most importantly, clear and supportive to learners. It takes a 'less is more approach' to assessment and feedback. Ackerman and Gross (2010) tell us that students become overwhelmed with copious feedback that is hard to penetrate. This means they are less likely to act on their feedback, which makes it less useful. The AFP does significant heavy lifting on the part of markers and course leaders by increasing student feedback literacy and setting their expectations around what good feedback looks like. This in turn allows students to meet the system halfway, as engaged learners. It gives markers a template to follow which helps them manage their workload within a marking team, and it gives helpful and supportive signposting to students about where to go and what to do next when they receive their feedback.

Implementation and Impact

The AFP has been implemented at Heriot-Watt University across multiple disciplines such as Mathematics, Computer Science, Engineering, Physics, Languages, Economics, and Psychology across all its global campuses in Scotland, Dubai and Malaysia. It has received positive evaluations from both staff and students with surveys over multiple years indicating high levels of satisfaction with the clarity, helpfulness, and perceived fairness of feedback delivered using the AFP. Additionally, uptake beyond the institution, in contexts including STEM disciplines at other UK universities, suggests a degree of adaptability across subject and institutional boundaries. Importantly, the AFP is intended as an Open Educational Resource (OER), aiming to support learners and educators across all disciplines to engage with the feedback process. The idea is that the AFP creates a system through which there is a form of teamwork that aligns all sides of the feedback dynamic: learners, educators and institutions (MacLaren 2026).

A Test Case: The University of Edinburgh's School of GeoSciences

The following describes the experience of using the AFP, in its accessible OER form, in two undergraduate courses at the University of Edinburgh's School of GeoSciences, in the academic year 2024-2025: GeoSciences Outreach and Engagement (GO&E) and Research Training for Geophysics (RTG). Neither course was restructured to incorporate the AFP, the protocol was adopted within the existing delivery structure. GO&E is a final year course, which is open to all undergraduates, but typically taken by students from the Schools of GeoSciences and Physics. In this course, students are given at least two supervisors — one member of academic staff, and one postgraduate tutor — whom they meet regularly. As outlined by Cross et al. (2022), this course stood out from 'conventional' courses in that the evaluation primarily focused on authentic assessment and student engagement. RTG, on the other hand, is a penultimate year course, mandatory in the Geophysics undergraduate degree. This course is more conventional in its assessment, as the students are assessed on two pieces of written work.

How Did the AFP Work Across Diverse Courses?

The assessment design in both courses remained the same, and the only change we implemented in the courses was the format of the system of feedback in order to align with the AFP. This was a useful test of the ease with which the AFP could be retrofitted to an existing assessment structure. However, as these courses are very different, the way we fitted the AFP into each course was adjusted accordingly.

In GO&E, there were six assessments spread throughout the course. For the first assessment, which was formative, supervisors provided the feedback verbally, structuring their approach using the 'Simplify the Message' dimension. After each assessment that followed, the supervisory team was focused on helping students to produce individual action points, building on the 'Encourage Action' dimension from the AFP. This process was repeated individually for each student on each of the assessments, thus making it a continuous process. The nature of the regular dialogue and action planning also allowed the 'Tune the Ear' content to be reinforced.

In RTG, there were two major assessments, both written: a literature review, and a group project report. The literature review, due in the final part of semester 1, was assessed as usual, according to four marking criteria (content; communication; presentation; and referencing) and comments were provided following the 'Simplify the Message' three comments structure. Here, the AFP was being rolled out in a more standard summative assessment sense, with dedicated class-time to 'Tune the Ear' and develop feedback literacy prior to the release of results. Once results and feedback were made available, we ran a further classroom

session to guide the students on how to utilise their feedback through a set of reflective exercises, in line with the 'Encourage Action' steps of the AFP. This session gave them time to write action points from their literature review to use in their forthcoming group project reports (the second assessment on the course), which enabled the students to use the AFP to feed forward into the subsequent assessment.

So, How Did It Go?

The AFP was well received in both courses. In GO&E, the course organiser noted two distinct benefits to embracing the AFP as the feedback system; the first underscores the fact educators should not overlook the power of motivational and affirmational feedback, and the second is aligning across larger and more diverse marking teams, a common challenge in modern higher education.

> "There were two main advantages to this approach. Firstly, the staff were encouraging reflection on well-handled skills, not solely skills needing to be developed and this promoted a balanced and inclusive approach to upskilling students. Secondly, the student feedback became much more consistent between staff and between assessment items." — Kay Douglas, Course Organiser of GO&E

Given the nature of assessment and supervision in GOE, it also felt natural to make the assessment feedback process more verbal and conversational, and empower the students to create action points based on both their strengths and weaknesses. It was also interesting to see that the system logic and structure of Tune the Ear, Simplify the Message, and Encourage Action could be translated to synchronous verbal communication as much as it could be used in asynchronous written feedback.

In RTG, there was an opportunity to include the students in the decision on using the AFP, and noting their opinions on it. After running a session on reflective practice and engaging with assessment feedback, the students were anonymously polled on how they perceive feedback currently in their degree and if they preferred this new approach. A total of 90% of the cohort welcomed using the AFP in 'most to all' of their courses, and a similar proportion felt more inclined to use their feedback if presented in this format. This gave us encouraging evidence that implementing a feedback system like the AFP could benefit nearly all courses. It is easy to incorporate alongside pre-existing marking rubrics and can streamline marking processes. Most importantly, it empowers the student voice, turning the feedback experience around from unidirectional to dialogical.

> "Only improvement would be more! If you'd be able to run a session earlier in the year OR in an earlier year" — 3rd year Geophysics student, enrolled in RTG

As suggested by one student in RTG, this could have even more impact if this approach to feedback was introduced from day one of the university experience. This could both empower the student voice in the assessment cycle, and standardise the student experience across multiple courses.

Conclusion

The AFP offers a structured and pragmatic response to persistent challenges in feedback practice. By aligning student expectations with staff capabilities, and by foregrounding simplicity and student agency, the AFP presents a viable and robust model for enhancing the educational value of feedback in higher education. Its emphasis on parsimony, consistency, and actionability makes it a valuable contribution to contemporary assessment design. As an OER, the AFP's resources are available and free to all. There are learner-facing resources and educator-facing resources. Feedback is a three-dimensional puzzle and our hope is the AFP helps manage that complexity. By taking a systems-level view, our intention is to create a sense of alignment between students' expectations, our approach as marking teams, and institutional norms around assessment.

Resources

- The Action Feedback Protocol (for educators)[135]
- Making the Most of Feedback (for learners)[136]

135 https://sites.google.com/view/theactionfeedbackprotocol/home

136 https://sites.google.com/view/makingthemostoffeedback/home

References

Ackerman, David S, and Barbara L Gross. 2010. “Instructor Feedback: How Much Do Students Really Want?” *Journal of Marketing Education* 32 (2): 172–81. https://doi.org/10.1177/0273475309360159.

Bevan, Ruth, Joanne Badge, Alan Cann, Chris Willmott, and Jon Scott. 2008. “Seeing Eye-to-Eye? Staff and Student Views on Feedback.” *Bioscience Education* 12 (1): 1–15. https://doi.org/10.3108/beej.12.1.

Boswell, Margaret. 2023. “Investigating STEM Pathway Students’ Perceptions of Reflective Practice: A Case Study.” *The Language Scholar* 7.

Cross, A., K. Douglas, Frederik D. Madsen, E. Zaja, C. Graham, and B. Auyeung. 2022. “Geoscience Outreach: What We Do, How We Assess, and Client/Student Reflections.” March 13, 2022. https://blogs.ed.ac.uk/teaching-matters/geoscience-outreach-what-we-do-how-we-assess-and-client-student-reflections/.

Higgins, Richard, Peter Hartley, and Alan Skelton. 2001. “Getting the Message Across: The Problem of Communicating Assessment Feedback.” *Teaching in Higher Education* 6 (2): 269–74. https://doi.org/10.1080/13562510120045230.

MacLaren, Andrew. 2026. “Feedback as Teamwork: Restoring Agency & Academic Quality Through the Action Feedback Protocol.” *Research Intelligence*, no. 166: 16–17.

Orsmond, Paul, Stephen Merry, and Kevin Reiling. 2005. “Biology Students’ Utilization of Tutors’ Formative Feedback: A Qualitative Interview Study.” *Assessment & Evaluation in Higher Education* 30 (4): 369–86. https://doi.org/10.1080/02602930500099177.

Weaver, Melanie R. 2006. “Do Students Value Feedback? Student Perceptions of Tutors’ Written Responses.” *Assessment & Evaluation in Higher Education* 31 (3): 379–94. https://doi.org/10.1080/02602930500353061.

Winstone, Naomi E, Georgina Mathlin, and Robert A Nash. 2019. “Building Feedback Literacy: Students’ Perceptions of the Developing Engagement with Feedback Toolkit.” *Frontiers in Education* 4: 39. https://doi.org/10.3389/feduc.2019.00039.

Winstone, Naomi E, Robert A Nash, Michael Parker, and James Rowntree. 2017. “Supporting Learners’ Agentic Engagement with Feedback: A Systematic Review and a Taxonomy of Recipience Processes.” *Educational Psychologist* 52 (1): 17–37. https://doi.org/10.1080/00461520.2016.1207538.

Winstone, Naomi E, Robert A Nash, James Rowntree, and Michael Parker. 2017. “‘It’d Be Useful, but i Wouldn’t Use It’: Barriers to University Students’ Feedback Seeking and Recipience.” *Studies in Higher Education* 42 (11): 2026–41. https://doi.org/10.1080/03075079.2015.1130032.

Assessment and Marking in an Introductory Programming Course for Physics Students

Ben Waugh

Department of Physics and Astronomy, UCL
b.waugh@ucl.ac.uk

Rebecca Chislett

Department of Physics and Astronomy, UCL

Louise Dash

Department of Physics and Astronomy, UCL

This informal case study describes how assessment is implemented in an introductory programming course for first-year undergraduates studying physics. It is not offered as a model to follow, but we hope someone developing a similar course (not necessarily in the same domain) will learn something from the approaches we have used, and how we have adapted them over time.

Syllabus and Course Design

Teaching Computing Alongside Practical Skills and Data Analysis

PHAS0007 Computing is one of three components of a 15-credit module on *Practical Physics and Computing* taken by all undergraduates in the Department of Physics and Astronomy at UCL in the first term of their first year, the other components being *Experimental Laboratory*, and *Data Analysis and Statistics*. After this introductory module, students take different computing courses depending on whether they are registered for a degree in *Physics*, *Astrophysics*, or *Theoretical Physics*.

The three components are taught separately, but the syllabus is aligned so that students can use computing to carry out data analysis, and apply both skill sets to the results of their laboratory experiments. However, the computing component is not just about statistical analysis: we also introduce basic computational physics, and teach students more general skills in building complex logic incrementally, structuring and commenting code clearly, and creating a scientific report in the form of a computational notebook.

Computing Classes

In the academic year 2025-26 we had over 300 students, divided into five groups because of the limited laboratory space, with each group having one 3.5-hour experimental laboratory session and one 3.5-hour computing class in each of the ten teaching weeks in the term. All students also have a weekly lecture on data analysis and statistics. Computing sessions take place in one of our experimental laboratories, with an individual Windows PC available for each student, although some choose to use their own laptop computer instead.

What We Teach

Our aim is not to teach coding as a general skill, or computer science as a discipline, but to enable our students to apply computational techniques to their work as physicists. We assume no prior knowledge of programming: while 87% of the 190 respondents to our pre-course survey said they had some experience of Python, 38% had not been taught programming at all and a further 21% “very little”. We use the Jupyter[137] Notebook application throughout the course, rather than an integrated development environment (IDE) or the more sophisticated but complex JupyterLab interface. We limit the extent of the computing concepts we attempt to teach in the course, using only simple data structures (lists and NumPy arrays but not dictionaries) and teaching students to define functions but not classes. We start with arithmetic operations, data types and variables, and importing and using existing functions. In the second week we introduce lists, arrays, loops, and defining functions. Later units cover plotting data, linear regression, summing series, and basic computational physics.

[137] https://jupyter.org/

Assessment Framework and Principles

The module has six summative assessment elements, a large number but one that reflects the wide range of skills that are covered in the three components. There are two assessments within the practical course (a formal report and a notebook assessment contributing 25% and 15% respectively to the module grade), one multiple-choice quiz on data analysis and statistics (contributing 15%), and three further elements within the computing component. Two of the computing assessments are extended "take-home" assignments: one mid-term data analysis assignment testing the concepts students learn in the statistics component as well as their programming skills, and a final computational physics task, contributing respectively 15% and 20% of the module grade. The remaining 10% comes from assessed weekly coursework exercises throughout the term.

Our approach can be described as *assessment for learning* (Wiliam 2011) in that "*all* assessment, within the overall package, should contribute to helping students to learn and to succeed" (Sambell, McDowell, and Montgomery 2012). Rather than creating separate learning activities and subsequent assessed tasks to measure performance, we try to design exercises that are interesting and develop understanding while also forming the basis of students' grades for the course.

UCL's regulations require the mark for each assessment component to be expressed as a percentage, and the overall module mark to be calculated as a weighted average of component marks. Within each component, module leaders can decide on how marks are decided, but all summative assessment must be *criterion-referenced*: marks are decided by checking submitted work against defined standards and not against the rest of the student cohort.

Continuous Assessment

Integrating Assessment With Learning in Weekly Sessions

The core teaching materials are a series of Jupyter notebooks that explain and illustrate the concepts we want students to understand, include code examples, and culminate in a programming task that guides students through the application of the week's material. Following consistent requests from students we also provide recorded screencasts talking through the content of the notes. In the class sessions, academics and postgraduate teaching assistants (PGTAs) offer individual help with understanding the examples and tackling the assignment. In 2025-26 each session was staffed by one of the two course leaders along with two further academics and two PGTAs. We try to guide students through the process of debugging their own problems rather than simply offering answers.

There are nine weekly *units* in the course: after the first five comes the UCL reading week, during which students work on the mid-term data analysis assignment, as well as assessments for other modules, but have no lectures or practical sessions. Then come five more sessions, although the last of these is not a unit in its own right but an opportunity for students to get help on the first stages of the final assignment and to catch up on any missed work.

The key aim of these tasks is formative: we explicitly encourage students to focus on the feedback they receive rather than on where they have "lost marks". Each unit contributes only around 1.1% (one ninth of 10%) to the module grade, but we have noticed that activities without any summative aspect tend to be skipped or at least rushed through by students. Some level of extrinsic motivation can be productive (Lin, McKeachie, and Kim 2003) despite the risk of over-assessment and an "assessment arms race" (Harland et al. 2015).

Automatic Marking of Multiple-Choice Questions (MCQs)

It is not practical to cover all taught material in authentic assessed tasks because of the excessive workload this would put on both students and markers. Each unit therefore also includes an online multiple-choice quiz, marked automatically, to help students check their understanding of the material in the notes before applying it. In the past this score contributed to the grade for the unit, along with the weekly programming assignment. In order to encourage students to use the quiz as a source of rapid feedback, we allowed unlimited retakes, with the highest score achieved being used. This still added complexity without any clear benefit, so we no longer use the quiz score in calculating the unit grade, but each student has to achieve a pass mark (typically 7 out of 10 questions) before they are able to submit the main assignment for the week.

Marking Programming Tasks In-Person

Prompt and Effective Feedback

This work is graded in person in class, primarily because we find this an effective way to give feedback: what we say to each student can be tailored to their own work and forms a dialogue which the student steers by their own enquiries and their responses to our questions. In contrast, written feedback is a one-way communication channel that takes significant staff time to impart guidance that may not even be read by the student, let alone meet their individual needs. This feedback is also timely: we encourage students to submit their work at least 30 minutes before the end of the session so that they can get feedback and a grade before they leave the room. This is not always possible, and some flexibility is important for students who may for

example miss a session due to illness, so we allow submission up to the day before the following session, and grade work in the following session or as soon as possible thereafter.

Integrity and "AI Proof" Assessment

An additional benefit of in-person marking is that we can check who we are talking to, and to some extent who has done the work. This is not the case for out-of-class marking of uploaded material. Integrity has always been a concern, but has become more of a talking point recently with the expansion of generative AI tools and the ensuing search for "AI proof" assessments. This year we have added an element to the rubric for each assignment that assesses the student's ability to explain their work or to make changes to their code when asked. This has proved useful in diagnosing student misconceptions and identifying areas where more practice is needed, as well as deterring misconduct. It has, inevitably, increased the time taken to mark each piece of work, so one of our aims for next year is to reduce the time spent on the aspects of marking that are more mechanical and less interactive. This might involve the use of automatic marking, based on unit tests or the Moodle CodeRunner[138] plug-in, or simply expanding the use of MCQs.

Organization

All course staff in the lab are responsible for both helping and marking students, with the balance shifting from the former to the latter over the course of the session. Staff move around the room and sit or stand next to the student whose work they are marking, using a tablet or laptop to assign marks in an online rubric in Moodle, our learning management system. We aim to mark students in the order that they have submitted, leaving them free to continue with further investigation or other work while they wait, and use a low-tech system to manage this: once they have submitted their completed notebook on Moodle, students are asked to write their name and desk number on a sign-up sheet at the front of the room. The marker crosses the next name off the list before going to the desk, checking the identity of the student, and then reviewing their work. Numerical marks are assigned by selecting a box for each criterion in the rubric, while feedback is given verbally. The process takes from around five minutes to fifteen or more, depending on how much discussion and explanation is needed. While this is time-consuming, we believe this is a productive way to use the limited contact time available and the expertise of the staff.

Assessed Computational Notebooks

Authentic Tasks

The two longer assignments are designed to be fairly authentic tasks in that they require the creation of a computational notebook including explanatory text and structured as a scientific report rather than simply a Python script. Students have more time to work on these, so the tasks are more ambitious.

Rubric Design

In any but the most simplistic rubric, there is a need for detailed explanation of each criterion and how to decide on the appropriate level for a given student submission (Brookhart 2018). It is often useful to illustrate commonly occurring responses or misconceptions, going beyond the level of detail that is useful for any given student, or fits into the rubric format used by Moodle. Hence we keep the criterion and level descriptions concise in the rubric itself, and provide markers with detailed marking notes explaining how they should be interpreted.

One change that we believe has been helpful, and has been positively received by markers, was to reduce the granularity of marking each criterion. Instead of giving a mark out of ten, we now define levels from zero (for no or minimal work) to a maximum of three or four. The intention, albeit it not achieved everywhere, is to define criteria for each level that build on those for the level below, so only work that meets the requirements for level two is then eligible to be considered for level three. This is intended to escape the sort of ambiguity that is common in mark schemes that make use of the word *but*, as in *has a well written conclusion but inadequate reference*s, leaving the marker to decide how to deal with a report that has outstanding referencing but a mediocre conclusions. This does take away the freedom of markers to use intermediate marks to reflect this uncertainty, but we hope this is outweighed by the benefit of reducing the time spent agonizing over small distinctions that make little difference to the final grade.

In practice each level is treated as a corresponding number of points, and the overall mark is calculated as the percentage of possible points awarded. A more sophisticated approach to weighting criteria and levels would be possible, but we see simplicity as a virtue, and additional complexity as unjustified in this case.

The marking takes time, with typically around 300 lengthy notebooks submitted. These are divided across ten or so markers, but it can take on the order of 30 minutes to mark each one, with the first few for each marker taking substantially longer. Limiting this workload, and ensuring the marking is completed within the four-week institutional deadline, are areas where we are continuing to focus effort.

Marking Fairly and Consistently

Marking is not conducted in person and is done anonymously in line with UCL's academic regulations, which does mean that there is no way to be sure whether a student has completed their own work, and what help (from other students, people outside the course, or

[138] https://coderunner.org.nz/

AI software) they have received. We use JPlag[139] to look for signs of collusion, but any indications of misconduct are investigated by the course leaders and the few cases that we pursue are decided by us using our own judgement, not by any automated process. JPlag has been successful in identifying cases where two students have submitted identical or very similar sections of code or text, but is not capable of flagging AI-generated work, where even the same tool can generate different output when applied multiple times to the same input.

While distributing the marking across a pool of markers limits the workload on any one staff member, it makes it challenging to ensure fairness (McConlogue 2020, chap. 6). The course leaders are required to second-mark a representative sample of work and take measures if discrepancies are found. However, it is hard to resolve problems after the marking is complete, short of re-marking from scratch, so we have put a lot of work into supporting consistency early in the process. The design of the rubric has been adjusted each year, trying to find a balance between well-defined and "objective" criteria, which leave little space for flexibility and professional judgement, and broader, more holistic criteria. The former tend to lead to a rubric with a large number of marking decisions to be made, albeit with each being more mechanistic and faster to process, while the latter mean fewer decisions but leave a lot of scope for different interpretations to affect marks unevenly.

Training and Calibration

No document can solve this problem and make marking *reliable* in the sense of giving consistent results across markers and students, except by making the assessment less *valid* by forcing us to use criteria that are artificially constructed and don't reflect the learning we care about. We can create a better process by introducing *calibration* sessions (McConlogue 2020, chap. 6) where we discuss our approach and ideally specific examples with the pool of markers. In a university context this is difficult because our markers have other commitments and are not all available at the same time for such a session. Hence we need to arrange multiple sessions and individual discussions, limiting the extent to which each marker can learn from all the others. An approach we currently use is for one of the course leaders to mark one or more student submissions, talking through their thought process as they do so, and responding to questions from other markers. This is particularly productive when the other markers are able and willing to mark one of their own allocated submissions in parallel, and raise points for discussion.

A Structured Data Analysis Assignment

The mid-term assignment asks students to carry out a data analysis task on a physics data set. It requires them to apply the concepts they have been taught in the data analysis and statistics course, as well as using Python code, calling appropriate library functions, to fit a straight line to the data and to interpret their results. This involves creating a report in the form of a Jupyter notebook, discussing the techniques they use. The instructions for this task are quite prescriptive, specifying the tasks to be undertaken and in what order, and giving specific questions that should be discussed in the report, such as to what extent the data are well described by a linear model.

An Open-Ended Computational Physics Assignment

The final assignment has a similar format to the mid-term assessment, albeit focussed on a computational physics problem rather than data analysis. Students are again required to investigate a problem and present their results and conclusions in the form of a Jupyter notebook. The most significant difference is that the final part of the task is fairly open-ended: students are led through the initial steps of an investigation, e.g. using Euler's method to predict the motion of a planet around a star, but it is up to them how they extend this, e.g. by tracking multiple bodies, investigating more complex orbits, or applying alternative integration algorithms.

The rubric is inevitably longer because the task is more extended and divided into multiple sections. Criteria like *correct code* are treated separately for each part of the task so that it is possible to deal consistently with cases where, for example, a student has successfully completed one section but not the rest of the assignment. As a compromise between explicitness and simplicity, we do apply some criteria (e.g. *code quality: comments*) to the notebook as a whole, and accept that in these cases a student can score well even with incomplete work.

The open-ended final investigation cannot be broken down into well-defined specific criteria, so we use only two: *thorough investigation* reflecting the ambition of the work attempted, along with its correctness, and *demonstration and discussion*, assessing how well the conclusions are explained and backed up by appropriate presentation of plots or animations. These criteria are marked out of four, and specific examples are discussed among the markers in an online forum in order to build a shared understanding of how to judge these.

[139] JPlag is an open-source sytem for source code plagiarism detection (Prechelt, Malpohl, and Philippsen 2000).

The Future

We continue to make minor adjustments to the course syllabus and assessment as issues arise, with an onoging focus on providing constructive feedback and fair marks to students while limiting the resulting staff workload. Possible future developments include more explicit use of frameworks such as standards-based grading (Pankratz 2026; Clark and Talbert 2023, chap. 5) or specifications grading (Clark and Talbert 2023, chap. 6).

Larger changes are also in the offing, both locally and in the wider world. A UCL-wide *programme excellence project* offers an opportunity to merge the existing module with others to form a more coherent 30-credit module taught over the full academic year. The worlds of education, research, and software development are also seeing major changes as generative AI techniques and tools become more powerful and pervasive. This has implications for the teaching of programming across disciplines that go well beyond the scope of this article but are discussed elsewhere in this volume (Evkaya et al. 2026).

References

Brookhart, Susan M. 2018. “Appropriate Criteria: Key to Effective Rubrics.” *Frontiers in Education* 3. https://doi.org/10.3389/feduc.2018.00022.

Clark, David, and Robert Talbert. 2023. *Grading for Growth: A Guide to Alternative Grading Practices That Promote Authentic Student Learning and Student Engagement in Higher Education*. Routledge.

Evkaya, Ozan, Hebatallah Shoukry, TJ Elmas, Laila Dabab Nahas, and Steven Watterson. 2026. “An Optimistic Outlook on Teaching, Learning and Assessment for Coding with the Emergence of Generative AI.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Harland, Tony, Angela McLean, Rob Wass, Ellen Miller, and Kwong Nui Sim. 2015. “An Assessment Arms Race and Its Fallout: High-Stakes Grading and the Case for Slow Scholarship.” *Assessment & Evaluation in Higher Education* 40 (4): 528–41. https://doi.org/10.1080/02602938.2014.931927.

Lin, Yi-Guang, Wilbert J. McKeachie, and Yung Che Kim. 2003. “College Student Intrinsic and/or Extrinsic Motivation and Learning.” *Learning and Individual Differences* 13 (3): 251–58. https://doi.org/10.1016/S1041-6080(02)00092-4.

McConlogue, T. 2020. *Assessment and Feedback in Higher Education: A Guide for Teachers*. UCL Press.

Pankratz, Elizabeth. 2026. “Developing a Skills-Based Framework for Assessments in Programming Courses.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Prechelt, Lutz, Guido Malpohl, and Michael Philippsen. 2000. “JPlag: Finding Plagiarisms Among a Set of Programs.” 1. Vol. 2000. Interner Bericht. Fakultät Für Informatik, Universität Karlsruhe. https://doi.org/10.5445/IR/542000.

Sambell, Kay, Liz McDowell, and Catherine Montgomery. 2012. *Assessment for Learning in Higher Education*. Routledge.

Wiliam, Dylan. 2011. “What Is Assessment for Learning?” *Studies in Educational Evaluation*, Assessment for Learning, 37 (1): 3–14. https://doi.org/10.1016/j.stueduc.2011.03.001.

Generative AI in Assessments: Before, During, and After

Lee Clift

Department of Computer and Information Science, University of Strathclyde
lee.clift@strath.ac.uk

Olga Petrovska

School of Mathematics and Computer Science, Swansea University
olga.petrovska@swansea.ac.uk

Introduction

You have likely seen the impact of Generative AI (GenAI) on your career as an academic, whether through journals discussing GenAI authorship or students attempting to get away with using Large Language Models (LLMs) to write essays for them. This chapter discusses the way to harness the power of GenAI to make your life as an educator easier, particularly in the context of planning, conducting, and marking assignments. We will discuss how you can use GenAI to write and refine assignments and solutions, how it can be encouraged as an exploratory tool by students, and how you can use the tool to speed up marking large class sizes. All of the following tips are LLM-agnostic; you can use them with any model you prefer, such as ChatGPT[140], Claude[141], or Copilot[142].

GenAI and LLM Basics

Throughout this chapter, we will go into detail regarding the use of GenAI and LLMs. Put simply, GenAI refers to computer systems capable of creating new content, such as text, code, or images, based on patterns they have learned from vast amounts of existing data. This allows them to answer questions and make new content for us, based on what already exists (such as information on the internet). LLMs are a specific type of GenAI focused on understanding and generating human language. In this chapter we use the terms GenAI and LLM interchangeably as all our examples are referring only to text-based interactions with GenAI.

It is important to highlight that our approach is LLM-agnostic. This means the techniques and prompts we discuss can be applied to whichever model you prefer, whether that is ChatGPT, Google Gemini[143], or Apple Intelligence[144]. While different models often have specific strengths, we have found them all to be highly capable of performing the assessment-related tasks we outline in the following sections. If at any point during this chapter you feel overwhelmed or confused, we invite you to pause, open up your LLM of choice (such as ChatGPT or Gemini), and simply experiment with it. Explore its outputs and try a few small tasks yourself. As with many practical skills, the best way to learn is through immersion and exploration — not just by reading about it here!

Before Assessment

Imagine you need to design an assessment — whether it is a coursework, a lab task, or an exam. Depending on the level of involvement required, designing this assessment can be a time-consuming process. Between thinking of a fun task, writing it out, devising solutions, creating rubrics, and formatting, it quickly adds up. GenAI can help you streamline this process as it offers several time-savers. Let's look at each of those in detail.

Using LLMs To Draft Multiple Choice Questions

Multiple choice questions (MCQs) are convenient in a variety of situations: you may want to quickly check content comprehension during a lecture, create a revision resource, or design a scalable assessment for a large cohort, which is free of grading bias. The idea behind MCQs is attributed to Wood and Thorndike and dates back to the 1920s (Engelhard 1988). One can design their own questions from scratch or refer to existing question banks. With the emergence of LLMs, educators have started exploring GenAI's capabilities to brainstorm ideas and accelerate the creation of MCQs. Some companies even developed custom-made solutions for MCQ generation and other teaching-related tasks, e.g. TeacherMatic[145]. Yet, these systems often require additional licenses that

[140] https://chatgpt.com/
[141] https://claude.ai/
[142] https://copilot.microsoft.com/
[143] https://gemini.google.com/
[144] https://www.apple.com/uk/apple-intelligence/
[145] https://teachermatic.com/generators-teaching/

your institution may not have. This, however, is not an issue and you still can generate MCQs using conventional general LLMs.

Tran et al. (2023) compared the capabilities of two OpenAI models to generate isomorphic MCQs, providing the Canterbury Question Bank[146] as a reference. Biancini, Ferrato, and Limongelli (2024) experimented with creating MCQs with LLMs, providing a prompt that addresses three basic components of MCQs: *stem* (context for the question), *key* (correct answer), and *distractors* (incorrect options). While the prompt they provide is specifically designed for history-related questions, it can still be used as a starting point for other subjects.

Practical Tip: Grounding Your Questions

To get the most accurate results, do not ask the LLM to generate questions from its general knowledge. Instead, provide it with your specific teaching materials, such as a lecture PDF, a PowerPoint, or a book chapter. This ensures the questions align exactly with what you have taught.

Example: MCQ Generation

Prompt example: Copy and paste the following prompt into your LLM after uploading your source material:

You are an assistant who doesn't make mistakes. If a reference model is presented to you, you follow it perfectly without making errors. Create a university-level quiz for [year x] students studying [subject] based on the provided file.

You must strictly adhere to the following format without any errors:
[Insert the question]
a) [Option A]
b) [Option B]
c) [Option C]
d) [Option D]

Correct Answer: [Insert the letter corresponding to the correct answer, for example: 'a)']

Source: [Write the exact line or passage from the provided file where the information for this question can be found.].

Note that you are allowed to modify only the parts within brackets ([...]) in the format provided.
Ensure that all four options are distinct.

Example output: Based on a lecture on Python programming, the LLM might produce:

```
Which keyword is used to
create a function in Python?
a) function
b) method
c) def
d) create

Correct Answer: c)


Source: "To define a function
in Python, we use the 'def'
keyword followed by the
function name." (Slide 14)
```

Reflection: The Importance of Human Oversight

Crafting effective prompts that specify the desired format of GenAI output and refer to relevant subject content is crucial for drafting acceptable MCQs. Yet, human oversight is always recommended, especially if the MCQs refer to less common programming languages. Educators should be the final authority on the quality.

Interestingly, though, a comparative study of human-crafted MCQs in programming versus those generated by AI has shown that the latter were actually more aligned to the learning objectives (Doughty et al. 2024).

Using GenAI To Write an Assessment Brief

Courseworks are usually rather involved assignments and require a detailed brief. The prompt "recipe" is similar to that of MCQs: provide a *context* (specify the subject, the learning objectives, the audience, and the level at which this subject is assessed), and determine a *format* (by either manually describing it or providing a sample document for reference). You can also use LLMs to help refine your prompt by asking them to serve as your guide — something some LLM's, e.g. GPT-5[147], already offer. Once you've got the brief, review and refine it with your human insight.

Practical Tip: Providing Examples

You can use GenAI to generate an assignment by following these steps:

1. Determine and define the key aspects: the assignment type, the concepts you wish to test, the level of study, and a potential theme. If you do not have a theme, you can ask an LLM for a suggestion.
2. Provide some key teaching material you have used (either as PDFs, PPTs or code snippets) to help "ground" your assignment. When doing so, ensure that the LLM you are using does not use your content for training and does not make it publicly accessible. Educational institutions often have special licensing agreements with specific LLM providers to protect institutional data, so it is worth checking whether your university has such an agreement and which provider it covers.
3. Provide a previous example (such as last year's coursework) that an LLM can use to format your brief similarly to your previous assignments, so they all look uniform.

[146] https://web-cat.org/questionbank/

[147] https://openai.com/index/introducing-gpt-5/

4. Do not be afraid to ask for multiple suggestions or request changes. Review and refine the LLM output, and then follow institutional moderation guidelines.

Example: Assessment Brief

Prompt example: Copy and paste the following prompt after uploading your materials:

Acting as a university module leader, draft a coursework assessment brief for [Level of Study] students in [Subject].

Use the provided [Teaching Materials] to ensure the task is relevant and use the [Previous Year's Brief] as a reference for the structure and tone.

The brief must include: A clear 'Task Description'. Specific 'Learning Outcomes' being assessed. 'Submission Requirements' (e.g., file formats, word counts). A 'Theme' for the project (If I haven't provided one, suggest three creative options).

Example output: Based on a Mobile App Development module, the LLM might produce:

```
Task Description: Develop a
cross-platform mobile
application using Flutter.


Learning Outcomes:
Demonstrate proficiency in
asynchronous programming and
UI/UX consistency.


Suggested Theme: A "Campus
Sustainability Tracker" where
students map recycling
points.


Submission: A GitHub
repository link and a 5-
minute video demonstration.
```

Reflection: The "human-in-the-Loop" Workflow

While GenAI is excellent at formatting and brainstorming themes, it lacks the context of your specific cohort's progress or institutional regulations. Some LLMs (such as GPT-4o[148] or Claude 3.5[149]) can even act as a guide, asking you clarifying questions to refine the brief further. Once the output is generated, always review it against your institutional moderation guidelines to ensure it remains a valid and fair instrument of assessment. Feel free to ask it questions, or to make modifications against your own style or institutional policies, such as late work submission policy.

Using GenAI To Design and Format a Marking Rubric

Once you have your brief written, you can get it to assist in writing a rubric. By stating the number of marks you want to award and some key learning objectives, you can quite easily create a marking rubric with discrimination between grades — you can even request this in various formats, such as a table or a standalone document.

Practical Tip: Building Step By Step

It is good practice to design your rubric in a step-by-step fashion. Here is a quick algorithm to do this:

1. Determine the key aspects that you want to evaluate — these will form the dimensions in your marking rubric. At this stage, you may prompt an LLM to provide suggestions based on specific learning objectives.
2. For each dimension, decide on the number of points it should be worth, making sure that they add up to a full grade across all dimensions.
3. For each dimension, prompt the LLM to break it down, depending on the number of points allocated for this dimension. Specify how fine-grained your rubric should be. Consider whether you want to base it on grade boundaries, the number of points, or work with ranges.
4. Review and refine the LLM output, then add it to the marking rubric.

Example: Marking Rubric

Prompt example: After uploading your Assessment Brief, use the following prompt:

Based on the provided assessment brief, create a marking rubric in a table format.

Use the following dimensions: [Insert Dimensions, e.g., 'Technical Execution' and 'Critical Reflection'].

Allocate [Insert Points] to each dimension. For each, provide detailed descriptors for the following grade boundaries: [Insert Boundaries, e.g., 70+, 60-69, 50-59, 40-49, and <40]. Ensure the descriptors show clear progression and discrimination between the levels.

Example output: For a 'Technical Execution' dimension (100 points), the LLM might produce:

```
Grade Descriptor:


70+ (Distinction)
Flawless execution. Code is
optimized, fully commented,
and follows all best
practices.


60-69 (Merit)
High-quality execution with
minor inefficiencies. Logic
is sound and mostly
commented.


40-49 (Pass)
Functional but unoptimized.
Basic requirements are met
with some structural errors.
```

Reflection: Refining the Rubric

While an LLM is excellent at generating "scaffolded" text, it often relies on generic adjectives like "good" or "very good." Review the output to ensure that a "70+" in your rubric specifically describes what a top-tier student in your field looks like. Additionally, you may not use terms like "good" and opt to use percentages or grade boundaries instead. Once refined, the LLM can also reformat this rubric into a standalone document or a table ready to be pasted into your Virtual Learning Environment (VLE).

[148] https://openai.com/index/hello-gpt-4o/

[149] https://www.anthropic.com/news/claude-3-5-sonnet

Using GenAI To Develop a Model Solution

Finally, we can request a model solution that incorporates both of these components. This will vary significantly depending on the assignment type; it works best for tests and less well for essays. A typical rule of thumb is that the larger the assessment, the worse the model answer will be. Regardless, these model answers can be used to quickly estimate what a solution may look like — we can even request multiple solutions from different ability levels! This will help us understand the typical LLM style for solving the problem and provide a better idea of when students overrely on GenAI output.

This can also serve an additional purpose — to test an LLM's capability to solve your coursework. This can be a good way to see how "LLM-proof" your assignment is, especially if the use of GenAI is prohibited. This can aid in either redeveloping aspects of an assignment or in developing an understanding of how an LLM may assist students in completing this specific assignment.

Practical Tip: Combining Tools

When asking for a model solution from an LLM, using both an assignment brief and a rubric can aid in getting better results. By using the rubric as a reference point, you can be specific about the model answer you are requesting, such as an answer that would receive 40% versus 60%, and quantify the differences.

Example: Model Solutions

Example prompt: Input your Brief and Rubric first, then use this prompt:

Act as a student attempting the provided assessment. Using the [Assessment Brief] and the [Marking Rubric] as your guide, generate two separate model solutions:

A 'High Distinction' (80%+) response that follows every criterion perfectly.

A 'Pass' (45-50%) response that meets the basic requirements but lacks depth in [Insert Specific Dimension, e.g., 'Analysis' or 'Optimization'].

Example output: For a programming task, the LLM might produce:

```
High Distinction: [Code which includes full error handling, modular code, and a detailed README.]


Pass: [Code which runs and completes the basic task, but lacks comments, uses "hard-coded" values, and has no error catching.]
```

Reflection: The "LLM-Proofing" Test

A general rule of thumb is that the larger and more open-ended the assessment, the more the model answer's quality will degrade. However, by generating these solutions, you can determine if your assignment is too easily solved by a single prompt. If an LLM provides a perfect 80%+ answer instantly, you may want to redevelop certain aspects, such as adding a reflective component or a specific real-world constraint, to ensure the assignment remains a true test of student skill.

During Assessments

While all the above uses of GenAI revolved around helping you to write an assignment, we should not forget that using LLMs effectively is becoming a desired skill (Petrovska, Clift, and Moller 2023). Hence, we need to prepare students for the real-world challenges, where businesses are proactively integrating LLMs into their processes. How do we do this? By offering them more authentic assignments that encourage them to explore different aspects of LLMs, students can learn to use these tools as helpers and not all-knowing oracles.

In our experience, students generally enjoy exploring GenAI as part of an assessment activity, provided it is not mandatory. Some students are ethically opposed to using GenAI; therefore, offering it as an optional assistive tool allows those who wish to engage with it to do so, without requiring participation from those who prefer not to use it. In cases where GenAI usage is required (such as reflective exercises — covered in the upcoming sections), providing examples for students to reflect without having to use the tool themselves can help mitigate any student concerns.

Tasks Around Interaction With GenAI and Critical Evaluation of Its Output

This can be anything from asking students to compare their own code with GenAI output or asking GenAI to provide several versions of a problem solution and encouraging students to determine which one is the best (Petrovska et al. 2024). Alternatively, students can be given a set of solutions — some human-written and some generated by GenAI — without being told their origin (Petrovska, Pearsall, and Clift 2025), and asked to identify the best solution and explain the drawbacks of the others. Such hands-on experience and exposure to GenAI's imperfections should, hopefully, build a more conscientious approach to using these tools.

Reflective Components in Programming Assignments

Historically, programming assignments have been evaluated based on whether the software could pass test cases and whether the code quality met a certain standard. From our informal discussions with other academics who teach programming, many begin to reconsider this approach. Understanding the code and being able to explain it becomes a more accurate criterion for assessing skill acquisition. If this resonates with you, consider incorporating various reflective components into your assessments, outside of typical code comprehension and understanding. Ask yourself: are students aware of the ethical and social implications of using GenAI? Are they aware and concerned about copyright and other legal implications? Are they observant and conscious about the impact of GenAI on their learning? Programming assignments do not have to be purely about programming, so feel free to add some extra tasks in the form of a reflective log, a video, or any other format that you find appropriate.

Allowing Unlimited (But Documented) Use of GenAI

In one of our experimental studies (Clift and Petrovska 2025), we have allowed students to use GenAI without restriction for their mobile application development projects. A word of warning: this type of exposure is not recommended for novice programmers, as their metacognitive skills and self-efficacy may not be well developed. They may not have acquired the required fundamental skills to fully appreciate and understand the benefits GenAI can offer or recognise its pitfalls. When applied to experienced programmers (for example, final-year degree students), the inclusion of, and unlimited access to, these tools can provide a constructivist learning environment in which students explore how the tools work and how to integrate them effectively into their own development processes. Results show that allowing GenAI usage during assessments leads to larger, more ambitious solutions being attained, while also encouraging students to learn about GenAI tools and their capabilities (Clift and Petrovska 2025).

Using GenAI as an Instant Feedback Tool

Alongside the explicit use of GenAI during a summative assessment, there is also potential to integrate GenAI into a more freeform, formative assessment process. This would allow students to explore and utilise GenAI during activities such as lab sessions, which are lower-risk and more casual than a typical summative assessment. One key example of this is CS50[150] — Harvard University's Intro to Computer Science class. The class is run both in-person and remotely, and is currently the most popular Intro to CS class in the world. CS50 introduced a GenAI model trained off their own resources called CS50 Duck[151] (Liu et al. 2024). CS50 Duck allows students, either remotely or in person, to seek instant help on problem sheets during formative assessments, while retaining the metacognitive benefits from reinforcing theory through problem sessions. This is achieved through the style of answers provided by CS50 Duck, including guards that prevent students getting a direct answer, instead helping them work towards the answer by asking questions, similar to those of a lab demonstrator or teacher.

Practical Tip: Prompting for an AI Tutor

Naturally, not all of us have access to the resources of CS50 or Harvard, and therefore, it can not be expected that we can build a system like CS50 Duck. However, by utilising an existing LLM, we can create something similar, through clever uses of prompts and by providing our taught content. This would allow us to provide an LLM to our students, enabling them to use it to solve problems rather than have it solve them.

Example prompt

Act as a Tutor for a computer science student.

I am working on [Topic/Problem]. My current logic is: [Insert Student Logic/Code].

Your Rules: Do NOT give me the correct code or the direct answer. Ask me a guiding question that helps me identify the flaw in my own logic. If I am stuck, give me a small hint related to [Insert Specific Concept or topic].

[150] https://pll.harvard.edu/course/cs50-introduction-computer-science

[151] https://marketplace.visualstudio.com/items?itemName=CS50.ddb50

After Assessments: Writing Feedback

Finally, we have received the students' summative assignments, and one of the most dreaded tasks is now here: marking! We do not endorse using GenAI to mark for you, although some educators have explored such possibilities (Banihashem et al. 2024) and several ed-tech companies are testing feedback generation capabilities of their GenAI solutions. One needs to understand that certain ethical and copyright concerns arise from submitting students' work to an LLM. Hence, we encourage continuing to mark the usual way, but utilising GenAI to help you create feedback that is detailed and constructive. With larger classes, providing all students with well-written, detailed feedback is a significant challenge and a time-consuming task. Here is a way to speed up the process while still keeping your feedback customised for each student.

Practical Tip: The "Feedback Persona"

Using GenAI, you can create a "Feedback Persona" simply by "teaching" a standard LLM your personal style.

The Setup: Open a new chat session in your LLM of choice. Paste 3–5 examples of your own past, high-quality feedback.

The Instruction: Provide your LLM with a description and some examples of your feedback style.

> This is my feedback style. It is encouraging but rigorous. I will provide you with bulleted marking notes, and I want you to turn them into two coherent paragraphs of feedback addressed to the student.

The Workflow: For each student, simply paste your shorthand notes. This keeps the feedback customised to the student's actual work while saving you the "blank page" fatigue of drafting 100+ individual paragraphs of feedback.

Example: Generating Feedback

Example prompt: Once you have established your style in the chat, use this prompt for each submission:

Using the feedback style I previously provided, transform these marking notes into a constructive feedback response for a student.

Marking Notes:

[Note 1: e.g., Great use of Python libraries, very efficient.]

[Note 2: e.g., Missed the requirement for error handling on line 42.]

[Note 3: e.g., Next time, try to modularize the functions.] ...

Ensure the tone remains [Insert Tone, e.g., Professional and encouraging].

Example output: The LLM transforms your notes into:

```
You have demonstrated a
strong grasp of Python
libraries in this assignment;
your implementation was
notably efficient and well-
structured. However, I
noticed that the error-
handling requirement was
missing around line 42, which
is critical for program
stability. For your next
project, I encourage you to
focus on modularising your
functions further to improve
code reusability.
```

Reflection: Ethics and the "Human-in-the-Loop"

When using a "Feedback Persona", data privacy is paramount. Never input a student's full name, ID number, or sensitive personal details into a public LLM. As mentioned earlier, some institutions may provide you with a more secure, local version of an LLM for safer use. Please read your institution's policy on GenAI and LLM usage before attempting this.

It is also important to consider what staff-generated materials are shared with commercial LLMs. Uploading assignment briefs, marking schemes, or model solutions to public systems may result in this content being incorporated into future model training data, depending on the provider's policies. This creates a risk that students could later access high-quality or even complete solutions via the same tools. Where possible, opt out of data sharing (sometimes referred to as incognito or private mode), use institutionally approved systems, or avoid uploading full solutions altogether.

Treat the AI-generated text as a "first draft" — always review it to ensure it accurately reflects your marking notes before sending it to the student. Transparency is also key; informing students that you use GenAI to help synthesise your handwritten notes into formal feedback builds trust and models professional GenAI usage. In our experience, students have been okay with this, and when they have challenged it, we have provided them with the raw feedback notes to show that marking has been done manually.

Conclusion

That's it; using those tips, you can adapt your assessment process to integrate this new tool into your practice, regardless of assessment type or field. We expect to see this field change and evolve. Many of the tips and prompts we have suggested can also be used to design simple GenAI agents — small, task-focused systems built on top of an LLM. In tools like Copilot these agents can be created in an accessible, streamlined form that effectively "pre-trains" the model to understand the structure, style, and type of output you want, without you having to repeat detailed instructions each time. Once you feel comfortable with the basics of using GenAI, exploring these lightweight agents can be a powerful way to simplify your workflow.

References

Banihashem, S. K., Nafiseh Taghizadeh Kerman, O. Noroozi, Jewoong Moon, and Hendrik Drachsler. 2024. “Feedback Sources in Essay Writing: Peer-Generated or AI-Generated Feedback?” *International Journal of Educational Technology in Higher Education* 21: 23. https://doi.org/10.1186/s41239-024-00455-4.

Biancini, G., A. Ferrato, and C. Limongelli. 2024. “Multiple-Choice Question Generation Using Large Language Models: Methodology and Educator Insights.” In *Adjunct Proceedings of the 32nd ACM Conference on User Modeling, Adaptation and Personalization (UMAP Adjunct’24)*, 584–90. New York (NY): Association for Computing Machinery. https://doi.org/10.1145/3631700.3665233.

Clift, L., and O. Petrovska. 2025. “Learning Without Limits: Analysing the Usage of Generative AI in a Summative Assessment.” In *Proceedings of the 9th Conference on Computing Education Practice (CEP’25)*, 5–8. New York (NY): Association for Computing Machinery. https://doi.org/10.1145/3702212.3702214.

Doughty, J., Z. Wan, A. Bompelli, J. Qayum, T. Wang, J. Zhang, Y. Zheng, et al. 2024. “A Comparative Study of AI-Generated (GPT-4) and Human-Crafted MCQs in Programming Education.” In *Proceedings of the 26th Australasian Computing Education Conference (ACE’24)*, 114–23. New York (NY): Association for Computing Machinery. https://doi.org/10.1145/3636243.3636256.

Engelhard, G. 1988. “Thorndike’s and Wood’s Principles of Educational Measurement: A View from the 1980’s.” Washington (DC): National Academy of Education. https://eric.gov/?id=ED295961.

Liu, R., C. Zenke, C. Liu, A. Holmes, P. Thornton, and D. J. Malan. 2024. “Teaching CS50 with AI: Leveraging Generative Artificial Intelligence in Computer Science Education.” In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education. Volume 1 (SIGCSE 2024)*, 750–56. New York (NY): Association for Computing Machinery. https://doi.org/10.1145/3626252.3630938.

Petrovska, O., L. Clift, and F. Moller. 2023. “Generative AI in Software Development Education: Insights from a Degree Apprenticeship Programme.” In *Proceedings of the 2023 Conference on United Kingdom & Ireland Computing Education Research (UKICER’23)*. New York (NY): Association for Computing Machinery. https://doi.org/10.1145/3610969.3611132.

Petrovska, O., L. Clift, F. Moller, and R. Pearsall. 2024. “Incorporating Generative AI into Software Development Education.” In *Proceedings of the 8th Conference on Computing Education Practice (CEP’24)*, 37–40. New York (NY): Association for Computing Machinery. https://doi.org/10.1145/3633053.3633057.

Petrovska, O., R. Pearsall, and L. Clift. 2025. “Assessing Software Engineering Students’ Analytical Skills in the Era of Generative AI.” In *Proceedings of the 9th Conference on Computing Education Practice (CEP’25)*. New York (NY): Association for Computing Machinery. https://doi.org/10.1145/3702212.3702223.

Tran, Andrew, Kenneth Angelikas, Egi Rama, Chiku Okechukwu, David H. Smith IV, and Stephen Macneil. 2023. “Generating Multiple Choice Questions for Computing Courses Using Large Language Models.” In *2023 IEEE Frontiers in Education Conference (FIE)*, 1–8. College Station, TX, USA: IEEE. https://doi.org/10.1109/FIE58773.2023.10342898.

Using Automated Marking in Programming Courses To Enhance Learning

Leila Shila Shafti

Computer & Information Sciences, University of Strathclyde
leila.shafti@strath.ac.uk

Joseph El Gemayel

Computer & Information Sciences, University of Strathclyde
joseph.el-gemayel@strath.ac.uk

In this chapter, educators with experience in automated marking for programming courses, particularly those aimed at non-Computer Science students, share their insights. The chapter introduces the concept of automated marking and provides a step-by-step tutorial on how to set up an assignment in Moodle using CodeRunner as the automated marker. We also share our practical experience of using automated marking and offer recommendations for educators who wish to implement it in their own teaching.

Introduction

In today's educational landscape, the demand for scalable, timely, and personalised feedback is higher than ever. Class sizes are increasing, digital and remote learning are now more popular, and educators are under pressure to maintain high-quality instruction while ensuring each student receives meaningful guidance. Traditional assessment methods, relying heavily on manual grading, often struggle to meet these demands, with feedback that can be delayed, inconsistent, or limited.

Automated marking uses technology to assess student work, enabling timely, consistent, and objective feedback across a wide range of assignments. This is particularly valuable in large classes and online courses, where manual marking would be impractical and immediate feedback is expected, and it also supports inclusion by providing equitable assessment for all students, regardless of background or accessibility needs.

In courses that teach programming, the need for automated marking is especially clear. Programming assessments are complex, and it is not enough for code to simply run or produce the correct output. Instructors often need to evaluate code structure, efficiency, readability, and students' comprehension of the code itself (Goodfellow et al. 2025). Manually assessing all these aspects is time-consuming, inconsistent, and prone to human error. Automated marking systems can manage these complexities, providing detailed, objective feedback that allows students to iterate on their solutions, understand mistakes, and develop stronger programming skills, all while maintaining fairness across large classes.

This chapter explores the practical implementation of automated marking, providing step-by-step guidance, and lessons learned from experience.

What Is Automated Marking?

Automated marking refers to the use of software systems to evaluate students' submissions without manual intervention. In programming courses, it enables the assessment of various aspects such as code correctness, efficiency, readability, and adherence to coding standards. Automated marking tools provide fast, consistent, and objective feedback, facilitating timely interventions and supporting large-scale assessments.

At the University of Strathclyde, we have delivered several programming courses to both Computer Science and non-Computer Science students. One of the main challenges in teaching programming is providing timely feedback to help students understand whether they are progressing in the right direction. Programming requires significant individual practice, during which students experiment with different approaches and develop their own code. However, students are not always aware whether their solutions are correct. This challenge is particularly significant for non-Computer Science students, who may have less prior experience with programming or computing.

Our experience shows that automated marking tools can help address this issue by allowing students to quickly identify errors in their code and refine their solutions, thereby improving their programming skills. Previously, Leila has used DOMjudge[152], an open-source tool designed to evaluate the correctness of program outputs. DOMjudge can be configured on a server to run submitted code against predefined test cases and determine whether the program passes all tests. This process enables students to identify bugs in their code and encourages iterative improvement. In addition, the automated testing environment introduces a game-like element that can increase student engagement, as students often find it motivating to run their code repeatedly until all tests pass.

There are several tools that can be used for automated marking in programming courses. One widely used tool is CodeRunner (Lobb and Harlow 2016), which is built on DOMjudge's framework for automatic evaluation of programming submissions. In addition to verifying correct outputs, CodeRunner offers enhanced features tailored for programming courses. It can provide automated feedback to help students improve their solutions. It integrates as a plugin for Moodle, allowing seamless incorporation into existing course materials. Moreover, the system is highly customisable, allowing instructors to implement more advanced assessments, including evaluation of code quality and other programming best practices. CodeRunner supports multiple programming languages, including Python 2 and 3, C, C++, Java, PHP, JavaScript (NodeJS), Octave, and SQL (SQLite3), with the flexibility to extend support to other languages. One significant advantage of using CodeRunner within Moodle is that it allows students' submissions to be processed on a university server, ensuring compliance with data protection regulations.

In the United Kingdom, several institutions have successfully adopted CodeRunner for automated marking and feedback. Goodfellow et al. (2024) at the University of Strathclyde have implemented CodeRunner alongside an in-house system, Browser Automated Marking (BAMjs), to assess Java, Python, and C programming languages across undergraduate years 1-3. The integration facilitated consistent and scalable assessments, and student feedback indicated positive reception to the automated approach. Similarly, Croft and England (2020) at Coventry University utilised CodeRunner for summative assessment in their first-year undergraduate programming curriculum, assessing both Python3 and C++14 code. The adoption led to improved key metrics, including faster feedback and reduced grading workload.

There are other tools for automated marking such as Autolab (Milojicic 2011), developed at Carnegie Mellon University, which automates grading for programming and data science assignments and provides detailed feedback on correctness and code quality. While these tools differ in features, flexibility, and integration with learning management systems, their common goal is to reduce instructor workload while providing timely and consistent feedback.

In this chapter, we focus specifically on CodeRunner, exploring its practical implementation, integration with Moodle, and benefits in programming courses. The following section provides a practical guide to implementing automated marking using CodeRunner, demonstrating how these features can be effectively applied in real teaching contexts. This is followed by a description of our experience using CodeRunner at the University of Strathclyde, and finally, the chapter concludes with key reflections.

Implementing Automated Marking

This section provides basic instructions on how to effectively set up an assignment in Moodle using CodeRunner as the automated marker. Before using CodeRunner, the plugin must be installed on Moodle.

CodeRunner can be easily customised for simple tasks, such as testing the output of a program, or for more advanced assessments, including evaluation of code style. While the tool is highly flexible, new instructors may require some initial learning to become familiar with its features and configuration options. To support this, this section provides straightforward examples and sample Moodle XML files that can be imported into Moodle. These resources help instructors quickly understand how to set up and use CodeRunner effectively, reducing the initial learning curve and enabling them to design assessments that meet the needs of their courses. For further details, the reader is referred to the CodeRunner documentation[153].

To use CodeRunner, an activity must first be added as a Quiz. Within the quiz, a new CodeRunner question can then be created. This will open the configuration page for CodeRunner.

When the student submits their code, CodeRunner tests the code with a set of test cases. The simplest test case is to define an input (if any) and the expected output. Then, CodeRunner runs the submitted program with the given input and compares the result of the program with the expected output. If the output of the submitted program does not match the expected output, the student will not receive any marks for that test case. Each question can have several test cases, with associated marks.

Therefore, the main step in setting up a CodeRunner question is to define the test cases (Figure 34.1). The most

[152] https://www.domjudge.org/

[153] https://trampgeek.github.io/moodle-qtype_coderunner/

important elements here are the **Standard Input** and **Expected Output**, which determine what the input and output of the program should be.

In addition to test cases, there are many adjustments that can be made, as can be seen in Figure 34.2. The most important ones — Question type, Precheck, Check and Penalty, Customisation, Template, and Support Files — are described in this section. Other options are also available in CodeRunner, and we recommend consulting the CodeRunner documentation[154] for detailed information.

Figure 34.1: Test cases in CodeRunner.

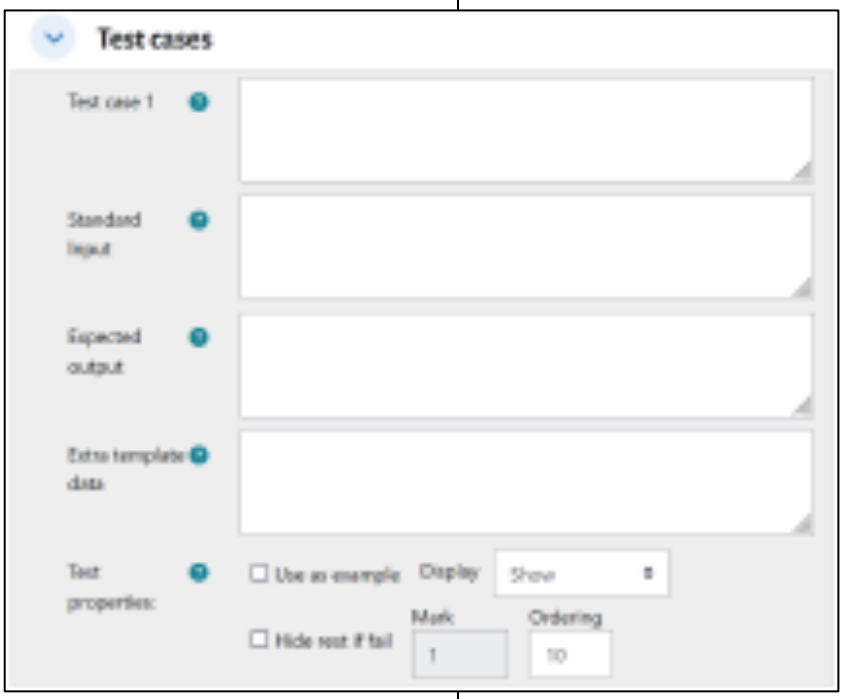
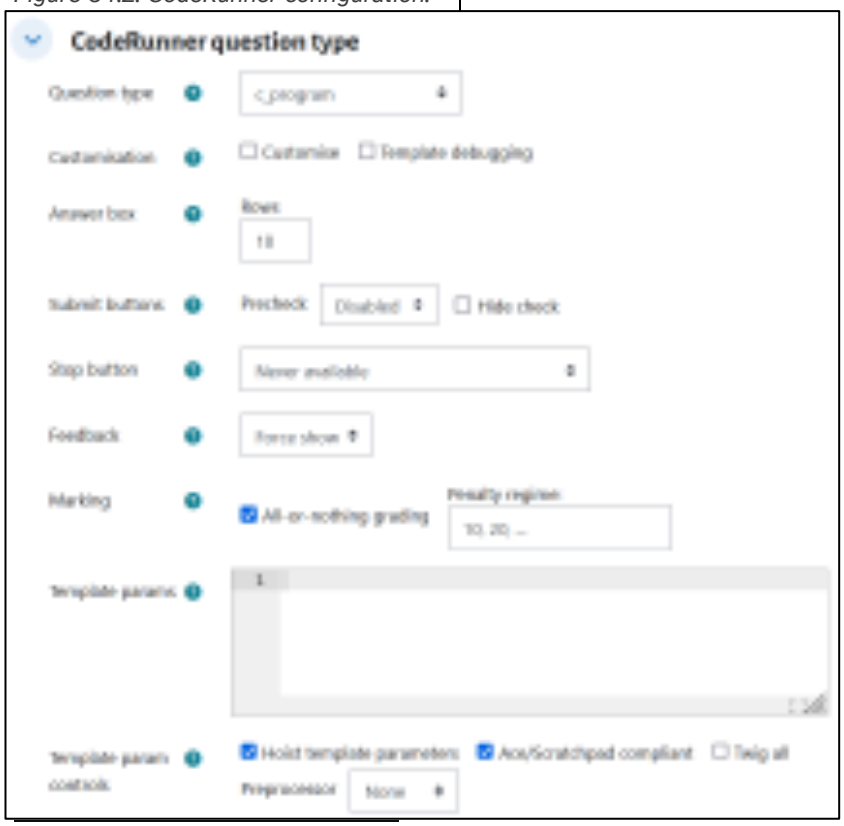


Figure 34.2: CodeRunner configuration.



Once the question is configured, it can be previewed and tested like any other quiz in Moodle. It is strongly recommended that, after the quiz is completed by students, the submission is reviewed for common errors and misconfigurations. This allows you to adjust the questions for the next cohort.

Question Type

Description: The Question type specifies the programming language used for the assignment. You can select one of the predefined languages from the drop-down menu. However, any language can be used once it is installed on the CodeRunner's server.

Recommendation: See the CodeRunner documentation[155] on supporting or implementing new languages.

Submit Buttons - Precheck

Description: Precheck determines whether the "Precheck" button should be available in the answer form (Figure 34.3). The Precheck button allows students to test their solution before it is marked, ensuring they have followed basic instructions, such as the required output format. This does not affect their mark.

Recommendation: Since CodeRunner compares the output of the program with the expected output, even a small difference may result in zero marks for the test. If the objective of the assessment is not the format of the output but the correct functionality, then Precheck can be used to test the submission against some basic input to ensure students are not penalised for the output format.

Check and Penalty

Description: In addition to Precheck, there is also a Check button in the answer form, as shown in Figure 34.3. This button allows students to verify if their answer is correct. A penalty regime can be defined to specify whether marks should be reduced if students change their solution after each Check. The Check button can be disabled by selecting Hide Check

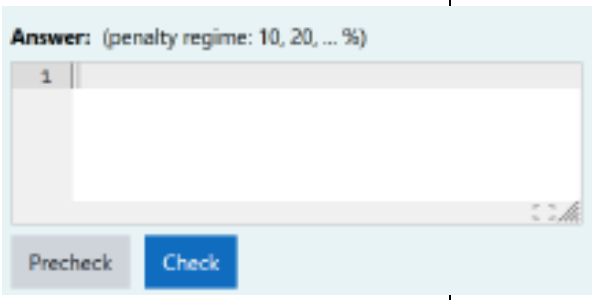
[154] https://trampgeek.github.io/moodle-qtype_coderunner/

[155] https://trampgeek.github.io/moodle-qtype_coderunner/

(Figure 34.2). Then, students would have only one chance to submit their code.

Recommendation: Using the Check button gives the student the opportunity to learn from their mistakes and improve their submission. However, depending on the assignment, you may want to hide this button and allow only one attempt.

Figure 34.3: Answer form



Customisation

Description: One of the powerful features of CodeRunner is that it enables customisation by selecting the Customise option (Figure 34.2). This will allow more complex customisation of the question, as shown in Figure 34.4. A useful option in customisation is Grading, which allows you to define how you want the output to match the expected output. It can be an Exact Match, a Nearly Exact Match (ignoring extra whitespaces and case differences), a Regular Expression, etc.

Recommendation: For most of our assignments, we use Nearly Exact Match, as we don't want to penalise students for extra spaces or case mismatches in their output. In some cases, Regular Expressions are used to focus only on a specific output without paying attention to other text in the output.

Template

Description: The template (Figure 34.4) allows more complex customisation of the question by defining a script to run for each test case or all test cases. The script defines how the student's answer should be evaluated. This enables more advanced evaluation of student submissions, such as assessing code style, enforcing specific language features, or analysing code structure. In this way, the template provides full flexibility to define the automated marker according to the requirements of the assignment.

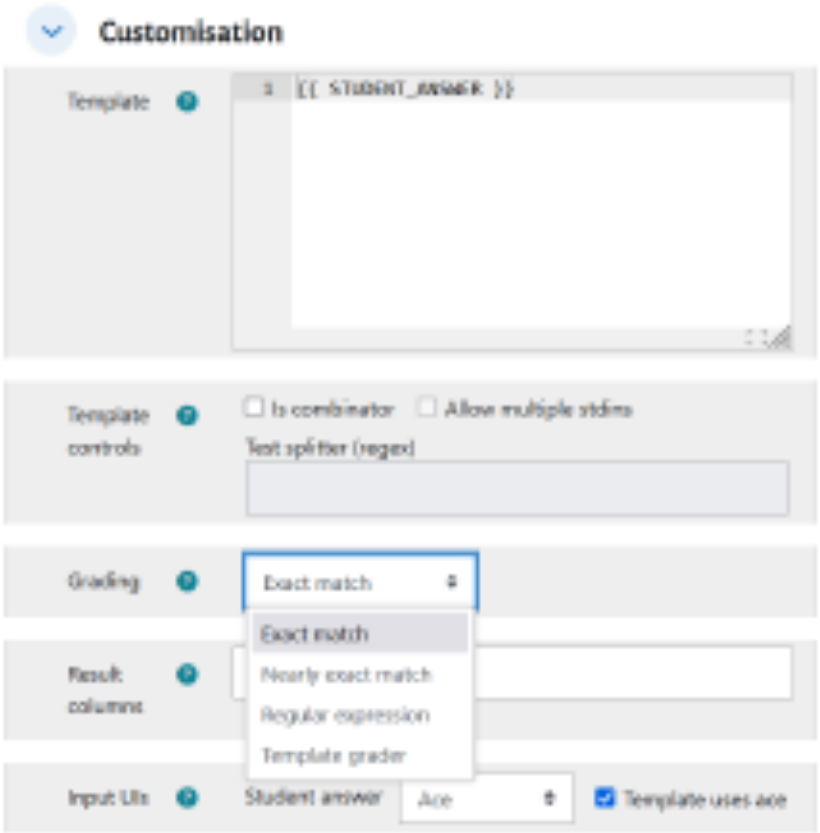
Recommendation: Use this customisation when the evaluation goes beyond checking for correct input and output, for example, when verifying that a student has used an array, a specific function, or other required programming constructs.

Support Files

Description: When additional files are needed to execute and evaluate a student's solution, they can be uploaded through the Support Files section of CodeRunner.

Recommendation: This section is useful for providing any supplementary files required for the assignment, such as additional C++ or Java classes, or even a Java parser for more detailed analysis of the student's code.

Figure 34.4: Customisation panel.



Our Experience

At the University of Strathclyde, CodeRunner has been integrated into several programming modules, including C, Python, Java, and SQL. Below are some examples of how it has been used.

Basic Input/Output Evaluation

CodeRunner is used to verify whether a program produces the correct output. Questions can be configured to require either a complete program or a specific function. When the output varies depending on the input, all relevant test cases are included in the question. See the Moodle XML file CodeRunnerBasic.xml[156] for an example.

Use of Specific Programming Constructs

In some cases, the goal is to ensure that students apply a particular programming feature. For example, in a loop evaluation task, it may be necessary to check that a student uses a while-loop rather than a for-loop. This can be enforced using the Template configuration. Depending on the programming language, the template script may be relatively simple, such as:

```
if 'while' not in
student_answer:
        raise Exception("ERROR:
You must use a while loop!")
```

The Template configuration can be significantly more complex. See the Moodle XML file CodeRunnerConstruct.xml[157] for an example.

Evaluation of JUnit Implementation

In some assessments, it was necessary to evaluate students' understanding of JUnit within Java programming. This was achieved by using a CodeRunner template[158] developed specifically for this purpose.

Evaluation of SQL Queries

CodeRunner makes it easy to design assessments involving SQL queries. The database required for the question can be created using SQLite and uploaded via the Support Files section of CodeRunner. Each query is then tested by comparing its output against the expected result. Since the order of rows in SQL query results is not guaranteed, students should either be instructed to use an ORDER BY clause on a specific attribute, or the template script can be configured to automatically sort the output of SELECT statements before comparison. See the Moodle XML file CodeRunnerSQLQueries.xml[159] for an example.

Evaluation of SQL Database Creation

CodeRunner can also be used to assess students' ability to design and create SQL databases. Test cases can be written to check whether tables are defined correctly, including the correct definition of primary keys, foreign keys, and data types. INSERT statements can be included in the test case to verify whether the student's tables enforce the required constraints. Since invalid inserts produce errors instead of standard output, a script must be added to the CodeRunner template to capture and display these errors as the student's output. See the Moodle XML file CodeRunnerSQLCreate.xml[160] for a simple example.

Conclusion

Automated marking provides several important benefits in the context of programming education. It can significantly reduce the workload for examiners and provide immediate feedback to students. Additionally, automated marking can transform the learning process into something more engaging and enjoyable. The instant feedback and challenge-response style of interaction create a game-like environment that encourages students to experiment, stay motivated, and actively engage with the material.

Automated marking is particularly valuable for students who find it difficult to ask questions or seek help during class. By receiving immediate and private feedback, these students can identify and correct their mistakes without needing to approach an instructor directly.

It is also especially beneficial for students from non-computing disciplines who are learning programming as a secondary skill. Without feedback, many of these students struggle to notice errors in their code, which can lead to misunderstanding. Automated marking ensures that even formative exercises provide clear guidance, helping students gain confidence and progress more effectively.

Finally, the high level of customisation offered by tools such as CodeRunner allows educators to design assessments ranging from simple input and output checks to more advanced tasks, such as evaluating specific programming constructs or features, code structure and style, or database creation. This adaptability makes automated marking a powerful tool for supporting diverse learning objectives.

156 /chapters/resources/C20_CodeRunnerBasic.xml

157 /chapters/resources/C20_CodeRunnerConstruct.xml

158 https://github.com/jackodsteel/coderunner-questions/tree/master/JUnitTesting

159 /chapters/resources/C20_CodeRunnerSQLQueries.xml

160 /chapters/resources/C20_CodeRunnerSQLCreate.xml

References

Croft, David, and Matthew England. 2020. “Computing with CodeRunner at Coventry University: Automated Summative Assessment of Python and c++ Code.” In *Proceedings of the 4th Conference on Computing Education Practice*. CEP ’20. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3372356.3372357.

Goodfellow, Martin, Andrew Abel, Konstantinos Liaskos, and John Levine. 2024. “Automated Marking in Undergraduate Programming Classes.” In *Proceedings of the 8th Conference on Computing Education Practice*, 13–16. CEP ’24. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3633053.3633060.

Goodfellow, Martin, Robbie Booth, Andrew Fagan, and Alasdair Lambert. 2025. “AutoMCQ - Automatically Generate Code Comprehension Questions Using GenAI.” In *Proceedings of the 30th ACM Conference on Innovation and Technology in Computer Science Education v. 2*, 737–38. ITiCSE 2025. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3724389.3731266.

Lobb, Richard, and Jenny Harlow. 2016. “Coderunner: A Tool for Assessing Computer Programming Skills.” *ACM Inroads* 7 (1): 114–18. https://doi.org/10.1145/2810041.

Milojicic, Dejan. 2011. “Autograding in the Cloud: Interview with David o’hallaron.” *IEEE Internet Computing* 15 (1): 9–12. https://doi.org/10.1109/MIC.2011.2.

Part IV — Case Studies

Pawel Orzechowski

Edinburgh Medical School, University of Edinburgh

Ozan Evkaya

School of Mathematics, The University of Edinburgh

Learning is exciting. That's because every next thing we learn is new, yet it somehow connects with what we already knew in unexpected ways. The same premise makes teaching exciting – each course, each lesson, becomes better and gets rediscovered each time we deliver it. As teachers we come up with new examples, or tools, or metaphors, to entertain (and educate) our students.

Unfortunately, it takes a lot of work to create or change our learning materials. It may take months from first having an idea to validating if it works with real students. Many of us take years to hone a specific teaching method or course format, as you've read in other chapters. But think about this cheeky idea: it's great to learn from our mistakes, but it's even better to learn from someone else's mistakes! Similarly, while it's inspiring to create fantastic new teaching experiences, there's also a lot of joy in reading about other people's adventures. Especially if they tell us about their journey, not just the end product – we can vicariously relive their struggle and their joy. We learn from their journey.

Case studies you will find below tell stories of challenging what it means to teach programming. They question where, how and why students can engage with code. You will find a diversity of course formats, delivery methods, subjects, and teaching styles. You will read about the nitty-gritty of small experiments and the epic logistics of big initiatives. We are very excited to take you on this journey, because... teaching is exciting!

Theme — Building Learning Communities and Resources

A Gentle Introduction to Coding (Orzechowski 2026)

Here's an example of how the love of coding can spread to people who never tried it before. Authors describe an approach to learning programming skills by building on student's previously existing thinking patterns, and how it could be used to run a course. A group of 20 brave social scientists spent a week of their summer to pair-program together, exchange metaphors, and add coding to their toolkits.

This chapter is for you if you are curious about how lofty, idealistic ideals (of 'gentle' teaching) can be translated into course design and teaching methods. It also includes a lot of practical advice about feedback gathering, course design, and the practicalities of running a bootcamp.

The Wild(Life) Side of Pair Programming (Sepúlveda 2026)

A fascinating story about a group of students from Scotland's Rural College who pair-analysed environmental data in Excel. Most of students did not join the BSc Wildlife and Conservation Management to learn coding or data analysis. But they will need those skills in their daily work of maintaining natural environments. The author describes a journey of searching and testing various ways to excite the class, which then culminates in cross-pollinating of ideas a buzzing classroom.

This chapter is for you if you want to be inspired by how a simple change in a teaching method can transform the classroom - especially when the teacher emphatically customises it for the specific audience.

Interdisciplinary by Design (Michielin and Otty 2026)

To create a sustainable training ecosystem it takes planning, skills and determination, as witnessed by the story of The Centre for Data, Culture, & Society (CDCS). This initiative was created within the University of Edinburgh to address the skills gap amongst non-technical researchers from diverse academic backgrounds. Over the years, a generation of scholars at different career stages used CDCS's training programs to enhance their digital literacy. The chapter showcases specific training materials and activities, from silent-disco to focused masterclasses.

This chapter is for you if you are looking for different training approaches that target scholars from non-technical backgrounds. You might get inspired by specific teaching materials, teaching methods or the very way in which CDCS serves its purpose. You'll also find useful insights if you're planning to start or rejuvenate a training programme in your own organisation.

Changing Civil Engineering Students' Mindset Toward Programming (Chien 2026)

Ponder this question: does being an expert in one area (e.g., civil engineering) change the way you program? The author describes what patterns they noticed among civil engineering students in terms of problem solving and code writing. You'll see how skills of connecting physical structures, logic and math get translated into code. Is their impact visible even in high level activities like code reuse, generalisation, and problem decomposition? It is not just about the differences in applications, but also about finding a golden middle ground between high-level and detailed, between applied and abstract.

This chapter is for you not just if you work with civil engineers, but also if you are an instructor in a field with its own specific ways of problem-solving. You might also enjoy pedagogical parallels woven from jumping between niche examples for engineering students, and high-level course design.

Lost in Translation (Konovalov, Michielin, and Yanina Bellini Saibene 2026)

Three educators combine their experiences of translating (and coordinating teams who translated) teaching materials from English into Ukrainian, Italian, Spanish and other human languages. They provide an actionable blueprint of how to start, run and maintain big translation projects, especially in the context of teaching coding. How do you make sure all translators are on the same page? What concepts do you translate and which do you leave in English? How do you ensure good quality of translations and motivate teams of translators (who are frequently volunteers)? This chapter has all the answers. Spoiler alert: it is all about process, source control, and synchronising expectations within the team.

This chapter is for you if you take part in (or manage) any translation projects, even if only tangentially connected to coding. Indeed, you will learn a lot about: coordinating teams; shaping quality-assurance processes; and future-proofing content. This chapter shows that anything worth doing, deep inside, is about project management and empathy.

References

Chien, Nguyen. 2026. "Changing Civil Engineering Students' Mindset Toward Programming." In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Konovalov, Olexandr, Lucia Michielin, and and Yanina Bellini Saibene. 2026. "Lost in Translation: Complexities and Good Practices of Translating Coding-Based Teaching Materials Across Multiple Human Languages." In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Michielin, Lucia, and Lisa Otty. 2026. "Interdisciplinary by Design: The Centre for Data, Culture, & Society Training Programme." In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Orzechowski, Pawel. 2026. "A Gentle Introduction to Coding: Narratives from a Programming Summer School for Social Sciences and Humanities." In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Sepúlveda, Ignacio. 2026. "Case Study from Scotland's Rural College: The Wild(life) Side of Pair Programming." In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

A Gentle Introduction to Coding: Narratives From a Programming Summer School for Social Sciences and Humanities

Pawel Orzechowski

Edinburgh Medical School, University of Edinburgh
pawel.orzechowski@ed.ac.uk

When people from humanities/social science start learning to code, they might feel alone in the dark end of the woods. But they are not, and you are not. As soon as you see programming as a storytelling exercise, your metaphors and narratives become breadcrumbs which will guide you on this transformative journey.

Disclaimer: this chapter is written like an annoying food blog

- It starts with a wishy-washy poetic blurb which you will probably scroll past anyway;
- followed by the actual recipe, details and process;
- just to finish with some practical recommendations the author may or may not have been paid to include...

(Also I will overuse punctuation (coders do that)). Enjoy!



Figure 36.1: Laptop stickers from the summer school, including the pair-programming proficiency badge

Coding Is Like Dancing, Right?

Everyone can dance, even if a little clumsily, even if we were never officially taught how to. And sure, we probably won't all star in a KPop music video, or dance a tango milonga with a partner. But we can definitely wiggle to the Barbie Girl tune, or to our inner music, and it will feel natural and fun. Dancing taps into everything we've ever learned to do with our bodies (like walking, jumping) and puts it in some context. It weaves a functional chain of input (music) and output (wiggling). It benefits from our strengths, and works with/around our challenges. Indeed, much of learning to dance is not learning new things from scratch (like juggling), but rather it builds on the things you already know how to do. Skills and thinking patterns needed for dancing (or coding) are just 'sleeping' in you, and you can combine them into this new ability.

What a wild metaphor, eh? It really holds its ground when you realise what Coding is. Coding IS NOT using a keyboard to type computer programmes in any particular programming language (python, R, javascript). Coding IS using your brain to understand a problem, split it into many smaller tasks and processes, and finally communicate it to another human and to a computer. Coding happens on a piece of paper, coding happens in your head.

> "I am such a huge fan of your teaching philosophy and skills. I really appreciated how inclusive your language and examples were, as well as your emphasis on kindness – I could really feel it in how you ran the course!" - Student feedback

> "At the end of it, I ended up being as excited about Python as about teaching." - Student feedback

Solving Sudoku Is Coding. Guessing Wordle Is Coding. Putting Plates in a Dishwasher Is Coding.

When you're problem-solving, you work around constraints and build a toolkit of mental skills and models. You chisel away at the problem in small chunks; you find small transferable mini-skills and patterns, that then you combine into bigger skills and patterns. We all do it all the time. The core component of this type of coding is that you do not solve one problem only once – you also learn to solve it better next time. And quickly you graduate to a higher level of abstraction – you see a bigger picture.

Like with learning to read, first we learn and read single letters (d-o-i-n-g), then we perceive letter clusters (do-ing), and finally words (doing). When I inhale the word "banana", I still understand it consists of letters, syllables, but I already ascended to a higher level of perception – the word feels yellow and sweet. It's the same with solving sudoku or wordle – you start seeing bigger patterns and rules. Similarly, when you solve puzzles using a computer as your sidekick (aka. you're coding), you first see small pieces (letters, brackets), then larger patterns (loops, functions), and finally you think in concepts and solutions.

That process of abstraction is familiar to most of us. Any learning, any sport, any skill works a bit like that. And the really magical thing is:

1. as we do things we get better at them (skill)

2. we also get better at getting better at other things (transferable meta-skill)

Imagine this: as you learn to play your first ever boardgame (e.g. chess, Scrabble, poker, or Settlers of Catan), you get better at that game. But as you learn to play your 10th boardgame type, you're also getting really good at the skill of learning boardgames. You start noticing common storytelling patterns



Figure 36.2: Instructors and some of the attendees of the CDCS summer school Gentle Introduction to Coding

in them, e.g. turns, points, odds or resources. You start snowballing skills of following any instructions or abstracting mechanics.

And before you notice, you get better at assembling IKEA furniture, or fixing your bike with a YouTube tutorial. You gain the meta-skill of learning to learn, and a scaffolding of patterns which will support your next lesson. A very similar thing happens with programming: you are NOT starting from scratch, and you get better at learning!

As You Learn Programming, You Will Use Skills and Thinking Patterns From Other Parts of Your Life

When you learn coding, just like any other skill, you anchor it in your skills, memories, metaphors and ways of thinking (Ambrose et al. 2010). You keep drawing on old patterns and constantly creating new ones. You solve more intricate puzzles and express those solutions in a very specific language. And as you master that new language you learn new nouns, verbs, acronyms and proverb-like shortcuts to achieve your goal quicker and more elegantly. You can imagine that this process gets easier as you do it more, right?

How to Apply This Practically in Learning?

Over the last 6 years I've taught thousands of people to code at the bootcamp and at the university. For each of my students this has been a journey of building on things that they already knew, their previous life experience and careers. But what skills do we all have which are relevant for coding?

I've seen students with experiences of hospitality or parenthood be great at pipelining, prioritising, just-on-time preparation, hundreds of small puzzles and numerous optimizations. Those who worked as book illustrators or language interpreters excelled at understanding someone else's vision and representing it in another format, managing process and dependencies, and doing it all within given time and quality constraints. I've seen my students combine tens of skills from different parts of their human experience, into how they UNDERSTAND coding. Everyone's journey to "getting" coding was different.

CDCS Summer School: 5 Days of Intense-Yet-Gentle Coding

"Gentle Introduction to Coding" was the week-long summer school for 25 brave students who joined me in June 2023. What connected most of them were two things: they came from social science and humanities; and they never learned to program before. For me this was my first time running a 5-day summer school, and I was anticipatory (aka anxious) of how it would combine my previous experiences of teaching (4-month intense coding bootcamps; semester-long university courses). This summer school was organised by the amazing team of Center for Data, Culture and Society[161] who also ran a more advanced summer school in the building next door. For all slides and coding exercises I used my teaching materials[162] from other courses. Below I will share with you some novel teaching methods I tried and how it went.

As well as seeing your peers' learning process, it is good to visualise your goals. Over the course of the week we were joined by people who use code at different stages of their careers in social sciences so that students see how diverse and impactful knowing how to code can be. Our keynote speaker Melissa Terras spoke about how coding ecosystems change in research; our guest speakers Alessia Calafiore, Bea Alex and Clare Llewellyn shared how they use coding in politics, literature and geography; while our teaching assistants Karim Rivera Lares and Chris Oldnall brought new perspectives from their work in psychology, maths and medicine.

Throughout the summer school, students submitted frequent 'relentless' feedback (also described by Alex, Llewellyn-MacRae, and Orzechowski (2026)) in the form of short bullet points about what resonated with them. Here are some of student comments combined:

- ⭐ I really liked the 'badges' approach – it allowed me to see the learning progress and easily identify the new blocks of knowledge acquired. They provide a bit of 'instant gratification' that a new level of knowledge was reached and that's a very motivating approach to teaching :) I helped to make big leaps I'd never have imagined possible over the five days.]
- ⭐ Careful guiding through the translation between common-sense and programmatic thinking in an approachable, easily understood manner (loads of storytelling, examples, metaphors, images, drawings, etc) helped me to absorb the new information a lot easier.
- ⭐ Effective pair format provided a supportive and collaborative atmosphere of learning together (and sharing skills across various professional and age stages). It was lovely to meet like-minded people and programming in pairs definitely helped overcome any previous obstacles I had when trying to learn code earlier by myself. It also helped to have support from the teaching assistants – Karim and Chris.
- ❓ I only wished to have a lot more time to practise the new skills, ideally over a longer period of time with the support from the teaching assistants/teacher. Maybe there is a scope for a follow-up course or meetup via CDCS?

"Pair Programming" Is a Collaboration Technique, Where Two People Share One Computer and Take Turns Typing on It

While the 'Driver' (person who types) talks out loud about what they are doing, the 'Navigator' (person who accompanies) is looking out for mistakes and asks clarifying questions. This allows the driver to focus on the challenge at hand, knowing that another set of eyes is checking what they are doing. At the same time the navigator supports and questions them. (I like to conjure an image of a little angel and devil duo sitting on the shoulders of a cartoon character). Every 15 minutes they swap roles, giving both participants the opportunity to practise both skills. This usually is repeated a number of times over a 2 hour session (Hawlitschek, Berndt, and Schulz 2022).

Did you ever notice that in the process of explaining your problem to someone you often find a solution by yourself? For example you cannot find something, and as soon as you ask your flatmate you find it ("oh, never-mind, I found that folder / wrench / salt. It was in front of my eyes all along"). Verbalising/spelling-out a problem (to someone, or to yourself) often facilitates solving it by yourself. Such structured 'talking to yourself' is very common in programming and we call it "rubber-ducking". That's because if there is no one there to listen to your problem, just find a rubber duck (a floating bath-toy type) and talk to it. And as you talk, your problem becomes clearer. In pair programming we have

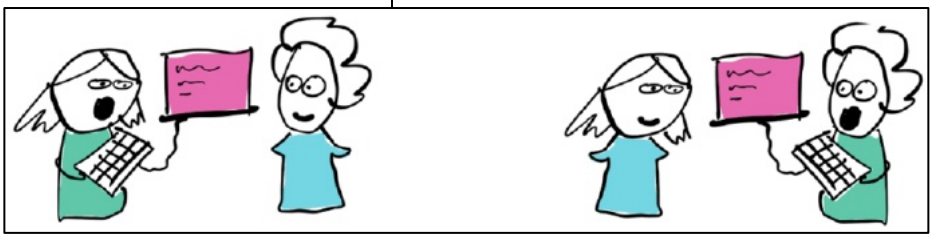


Figure 36.3: When two students use pair-programming: 'the driver' (left) is a person who types, 'the navigator' (right) is their companion, as they solve a problem together.

[161] https://www.cdcs.ed.ac.uk/

[162] https://www.codestorytelling.com/

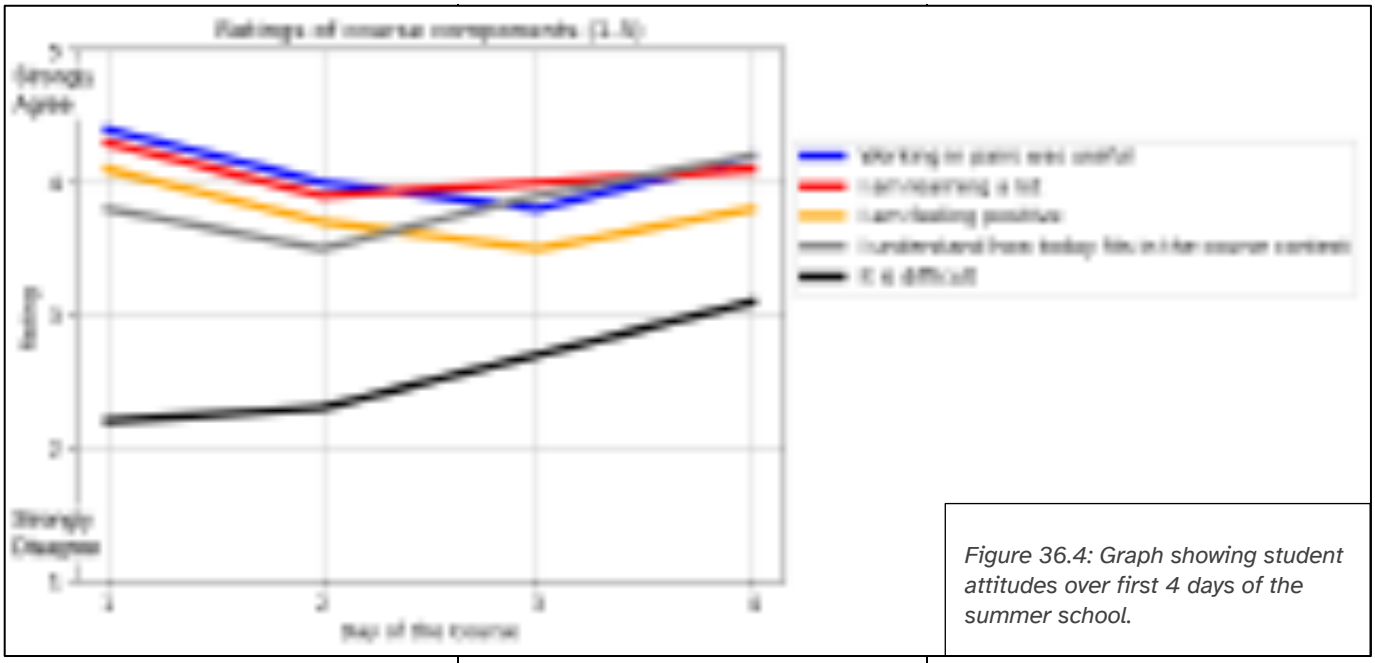


Figure 36.4: Graph showing student attitudes over first 4 days of the summer school.

something even better – a partner who can not only listen, but also asks questions. Often they are at a different level than you, so questions can be unexpected and enlightening. Through this practice students learn transferable skills of how to talk, listen and negotiate in a collaborative environment.

Cross-pollinating ideas in pair programming works surprisingly well in diverse groups[163]. Each partner brings their own background, metaphors and ways of understanding, while also experiencing someone else's ways. All learning during the summer school happened with partners, who we changed every 2 hours. That meant that the diversity of skills and backgrounds worked to our advantage, as it led to spreading of good practice and also allowed people to meet with each other.

If you'd like to know more about Pair programming, join our special interest group[164] or read more about how to introduce it into your classroom in other chapters of this book (Orzechowski, Blankinship, and Banas 2026; Desvages et al. 2026).

Badges: Collect Skills Like Pokemon!

I find it easier to learn something new when it is separated into manageable chunks, especially when I understand how each chunk fits into a bigger picture. I also love it when assembling all the pieces feels personal, fun and challenges how I see things. That's why I resonated so deeply with Threshold Concepts (Meyer and Land 2003) (thank you PgCAP[165]!) – a theory that some knowledge cannot be unlearned and is transformative (like crossing a door threshold). That's why all of the summer school content was separated into 15 Badges (achievements, blocks, sections). Each badge was a 1.5-hour-long chunk of knowledge/skill and each built on top of all previous achievements. To collect a badge students participated in a 20 minute interactive presentation from the teaching team, followed by an hour of pair-programming, and a few minutes of self-reflection.

Figure 36.5: Examples of badges that students could achieve during the summer school.



[163] By the way, 'pear' is spelled differently than 'pair', but did you know that if two pear trees exchange pollen they will have more and healthier fruit?

[164] https://pairprogramming.ed.ac.uk/

[165] https://institute-academic-development.ed.ac.uk/learning-teaching/cpd/postgraduate-certificate

Figure 36.6: Components of each badge are: slides, coding exercises and self-reflection mini-diary.



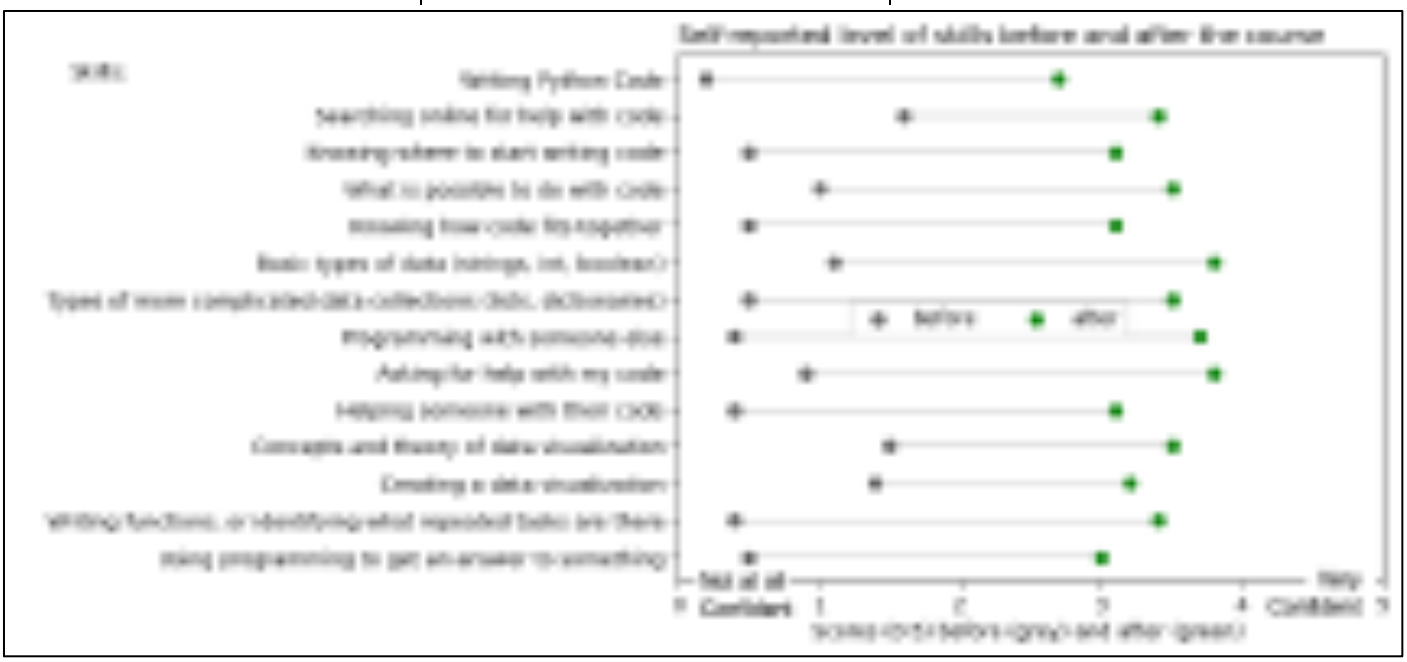
Have you ever tried writing a "gratitude journal" at the end of the day, by listing things that you are grateful for? It consolidates experiences you had, frames them in positive ways and enables incorporating them into our fabric of experience (Bono et al. 2020). To reflect on the personal and transformative component of learning, students ended each badge by writing a 4-sentence-long self reflection. We used the "3 stars and 1 wish" method from primary schools, where everyone lists 3 things that spoke to them personally, and 1 thing they wish they understood better (Llewellyn, Orzechowski, and Alex 2020). Every 2 hours, every student wrote that mini-reflection. Externalising thoughts is designed to help everyone to remember and encode what they learned, but also allows the instructor team to identify parts of content which need revisiting.

We leaned into our inner children not just during the reflective mini-diary, but also during the interactive parts of the presentation. Did you notice that it can be surprisingly challenging but also joyful to wonder "How would you explain this to a very smart 8 year old?". It was great to see how kind and clear people's descriptions are when they imagine talking to a child. We also played "I'm not a native speaker, what does this word mean?" which was particularly joyful since me and many students did not grow up speaking English. It is very surprising how technical words have their roots in everyday language, and exploring those roots does make it easier to integrate the jargon. For example "index" points at things, (just like an index finger) and "key" unlocks one value (just like door key).

> "I would recommend this course to anybody with even the tiniest inkling of interest in coding, even if (or maybe especially if) they can't quite imagine how it could help them in their work or research." - Student feedback

> "I had been looking for a course like this for a while, but found the tutorials and online courses too complicated, and self-learning too overwhelming starting from zero. [...] I had been 100% dreading getting to the social media analysis part of my PhD – after this course, I'm really looking forward to getting to it!!" - Student feedback

Figure 36.7: Graph showing students' self-perceived improvement in terms of skills before and after the summer-school, on scale from 0 to 5



As Promised: Some Practical Top Tips for Running Similar Courses

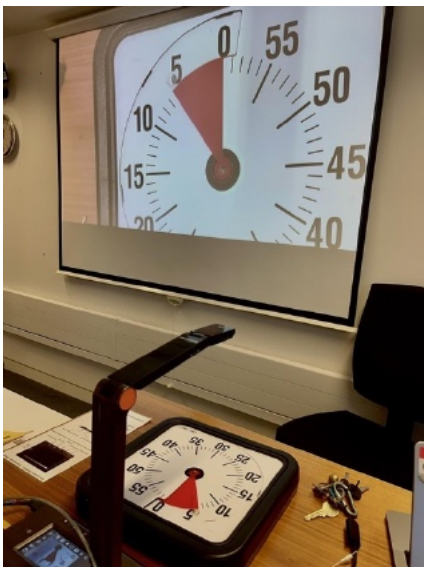


Figure 36.8: 'Time Timer' is a large clock designed for primary schools which we used to indicate when driver and navigator should switch roles.

1. Use "Time-timer" for visually indicating 15 minute periods between switching seats in pair programming (I used a big one and a small one).
2. Use 10 meter power extension cords with 5 plugs each to not daisy chain cables – lay them out in front of the rows of tables and tape down to the floor with lengths of gaffa tape. Insist people plug-in in front of their table, not behind, so that you don't trip while walking between desks.
3. Find a room with rows of tables (not a 'lecture theatre' style), so that you can comfortably reach each student's computer when helping with their practicals.
4. For presentations big screens are brighter than projectors, but much smaller, and also tend to be mounted lower on the wall. A big projector usually projects higher so it will make it easier for people at the back to see the slides. If you're in the room with a low-mounted big screen, you might have to edit your slides to have nothing important at the bottom of the slide (bottom section of each slide was invisible to students sitting at the back).
5. Go to your lecture room on the day before to check projectors, chairs, cables. In our room the previous room users unplugged all AV equipment for some reason, but luckily we noticed in the morning and called the wonderful tech-support team who fixed it on time.
6. For quick questionnaires and feedback use Microsoft Forms available to all staff and students (not the ones where you need to be logged-in). They also present results in a quick nice set of graphs.
7. Chrome browser has the ability to share the currently displayed page as a QR code (next to the star on top right), which is useful when telling people to look at something on their phones, e.g. quick feedback forms.
8. Use a cloud computing platform if possible. Noteable[166] is a fantastic online coding environment free for our students, built by the Edina team at the University of Edinburgh. With it you can start coding in Python/R in minutes, and it works great with GitHub.[167]
9. Sharing notes on GitHub is quick and convenient. Instructors can add new notes, and the class 'pulls them' into their computers in seconds.
10. Try to have at least 2 teachers in the room, so that the team can take breaks without abandoning students.

[166] https://noteable.edina.ac.uk/

[167] If you ever had to troubleshoot why Python or R does not work in a classroom of 150 students using 3 operating systems in 6 foreign languages... you understand why it is precious to be able to skip this step and code immediately with no setup ('out of the box'). Especially for online or large classrooms.
We've seen two solutions to this challenge: at codeclan we bought all students the same model of a MacBook (super expensive, but very fancy); at the University of Edinburgh all students have access to an identical and powerful 'virtual computer' via their web browser (cheaper and scalable).

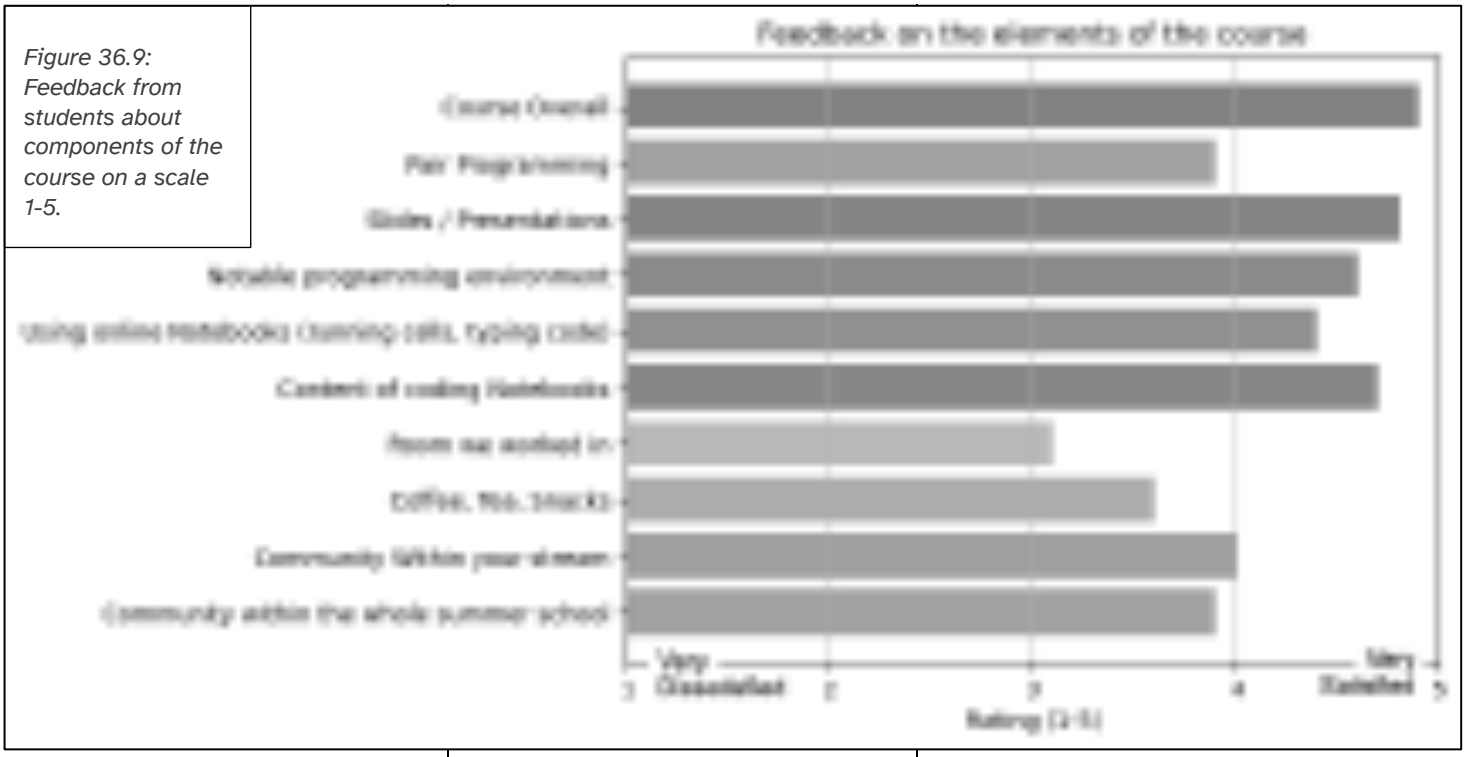


Figure 36.9: Feedback from students about components of the course on a scale 1-5.

This Is the Beginning of a Beautiful Friendship

To sum it all up, our summer school combined a number of teaching methods, which did resonate with the participants. Splitting content into Badges enabled us to scaffold learning, and be very agile about which parts to skip, and which to expand, following what the group wanted. Practical exercises all involved pair-programming and constant peer-feedback for cross-pollination of skills. Frequent reflections with 3 stars & 1 wish and classroom discussions created a feedback loop between the students and the teaching team (shout out to Karim and Chris, my amazing teaching assistants). Additionally, since most participants were early career academics/students, their feedback often mentioned not just learning to code, but learning how to run a well-designed, modern and interactive course.

References

Alex, Beatrice, Clare Llewellyn-MacRae, and Pawel Orzechowski. 2026. “Learning Together Across Modes: Online and on-Site Pair Programming in a Fusion Course.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Ambrose, Susan A, Michael W Bridges, Michele DiPietro, Marsha C Lovett, and Marie K Norman. 2010. *How Learning Works: Seven Research-Based Principles for Smart Teaching*. John Wiley & Sons.

Bono, Giacomo, Susan Mangan, Michael Fauteux, and Jason Sender. 2020. “A New Approach to Gratitude Interventions in High Schools That Supports Student Wellbeing.” *The Journal of Positive Psychology* 15 (5): 657–65. https://doi.org/10.1080/17439760.2020.1789712.

Desvages, Charlotte, Brittany Blankinship, Umberto Noè, and Pawel Orzechowski. 2026. “Structured Group Work with Assigned Asymmetrical Roles and Switching: Lessons from Pair Programming Across Disciplines.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Hawlitschek, Anja, Sarah Berndt, and Sandra Schulz. 2022. “Empirical Research on Pair Programming in Higher Education: A Literature Review.” *Computer Science Education* 33 (3): 400–428. https://doi.org/10.1080/08993408.2022.2039504.

Llewellyn, Clare, Pawel Orzechowski, and Beatrice Alex. 2020. “Teaching a Text Mining Bootcamp in Lockdown.”

Meyer, Jan, and Ray Land. 2003. “Threshold Concepts and Troublesome Knowledge (1): Linkages to Ways of Thinking and Practising Within the Disciplines.” *Princeton: Citeseer*.

Orzechowski, Pawel, Brittany Blankinship, and Kasia Banas. 2026. “Where Do i Even Start with Pair Programming in My Classroom? A Conversation with Seasoned Practitioners.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Case Study From Scotland's Rural College: The Wild(Life) Side of Pair Programming

Ignacio Sepúlveda

Oatridge College, Scotland's Rural College
ignacio.sepulveda@sruc.ac.uk

Natural habitats

I've been teaching the Wildlife and Conservation Management[168] programme at Scotland's Rural College for almost 6 years, including modules around climate change, marine habitats, wildlife conflicts, survey methodologies, and map analysis. My groups are typically filled with students of all academic backgrounds and ages, united by a deep desire to protect our natural environments. However, most arrive with little to no IT skills. This is a significant gap, as I consider these abilities vital for any professional in the field, even those who envision themselves as solitary park rangers in the highlands or in the remote islands of Orkney or the Shetlands.

I strongly believe that learning how to code can have a huge impact on how these professionals improve and streamline their work, whether analysing climate data or sharing their findings about endangered species with the world, basic coding skills should be part of their day-to-day toolbox.

As my background has always related to physical sciences and programming, especially around R[169] and QGIS[170], I felt the need to show my students that world. A world normally hidden to them under the idea that only 'computer skilled' people are allowed to explore it. We were about to demolish that myth together.

Always Changing

This IT module has evolved quite a lot from its initial structure. During my first year teaching it, the module had a more basic and general approach; it served its purpose as a "work-for-all" type of module, but I realised that by the end of term the students were craving more skills to make their lives as future ecologists easier.

Over the next couple of years, I added a few workshops on advanced Excel and Word as an extra feature, and this year, for the first time, the students will also have a brief introduction to R and QGIS.

A Leap of Faith

Over the past two years, I've been experimenting with new teaching techniques for my second-year students, having them share roles while tackling Excel data analysis or collaborating on map creation in QGIS. Looking back now, I realise I was using pair programming (Desvages et al. 2026) (two students working together on one computer) all along, unaware of its formal framework or established structure. It had emerged from my imagination during those late Sunday night planning sessions, born out of necessity and intuition. That initial gut feeling pushed me to learn more about pair programming, and I was ready to apply it. I wanted to show my students that programming isn't just for coders; it's for them too, demonstrating that when we work together, share our thinking, and build on each other's strengths, even the most intimidating technical challenges become accessible.

[168] https://www.sruc.ac.uk/course-catalogue/wildlife-and-conservation-management/bsc-hons-wildlife-and-conservation-management/

[169] R: open-source software for statistics and data analysis.
[170] QGIS: open-source software for mapping and spatial data analysis.

The Challenge Begins

My first-year students had already a gentle introduction to Excel, mostly through a few easy-to-follow examples that they then replicated in their own computers. A good start? Yes. Enough for their current needs? Not quite. That was when pair programming emerged as a good idea to try - for the first time in this module.

A week has passed and I welcome my students once again, a group of around 25, and present the challenge. I'm transparent about the difficulty level; a significant portion of them have never used Excel or even approached a laptop this way before. I distribute specific instructions and explain what pair programming brings to the table.

We'll work in pairs. One student will be the **driver**, controlling the main PC and executing tasks, while the **co-pilot (or navigator)** helps problem-solve and strategize. The catch? Roles swap every 15 minutes, guided by a giant clock projected on the wall.

Figure 37.1: Students in their pair programming roles



Get Real Messy Data and Figure It Out. Together.

This time, they had to download a real dataset from the MetOffice: climate data from Scotland in a strange-looking format (.txt). It involved conversion, structuring, and cleaning all at once. I could see them struggling with this. Some groups were trying different options, while others were looking for answers on websites or asking me questions. I could see how different strategies arose, but most importantly, each pair was using their different skill sets to solve this first part. Some of them almost gave up, while others were thriving and moving into the analysis stage.

Fifteen minutes passed and the roles changed. There were some chuckles to hide the despair, but overall, I saw optimism and the spirit of cooperation. Most of them went through the first section and started their analysis: sorting and filtering columns, organising data, using formulas. I tried to explain that, although basic, the use of those formulas, especially the newly introduced "IF" and "AND/OR", are fundamental for any programmer, and that these short formulas are actually lines of code. I could see some sparks in their eyes.

A few more 15-minute rounds cycle by. Most groups progress steadily, completing their analysis and beginning their presentations, creating charts, exploring pivot tables, extracting meaningful insights. Others, however, are visibly exhausted. The dynamic of this challenge is demanding, putting real pressure on them. Some are still wrestling with formulas, while others are identifying inconsistencies in the original dataset. I do my best to guide and support these struggling groups, offering hints without removing the challenge entirely.

Figure 37.2: Digital clock on the whiteboard showing the 15 minutes intervals.

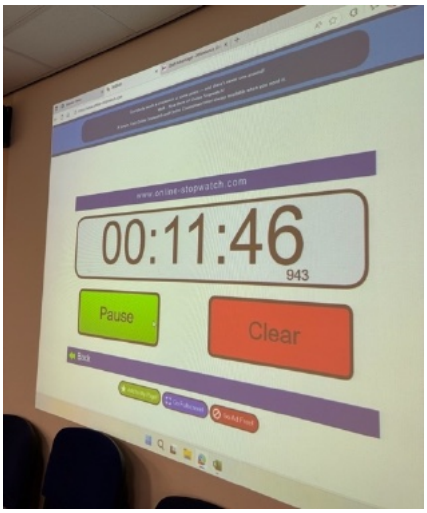




Figure 37.3: Group of first year students in the middle of their pair programming challenge.

Feelings! All the feelings!

Time's up! That was it for today's challenge. I could notice some tiredness mixed with happiness. Some students were proud of completing all the instructions, while others were still a bit confused. I was clear that this had been a real challenge, and these are never easy. Still, I told them how impressed I was. To be honest, I wasn't expecting this level of courage and commitment with a new activity like this one, which requires cooperation, trust, and agility like no other.

Just as the last group prepares to leave, one student turns back. "Ignacio, could you beat this challenge yourself? Why don't you time it and show us next week?" I wasn't expecting that, but I couldn't resist. "Challenge accepted," I said, laughing. "I'll record my attempt and see if I can beat my personal record for the next Excel championship."

Walking back to my desk, I realised how much this small experiment has transformed the atmosphere of the class. Pair programming converted what could have been a routine IT lesson into a shared experience, a moment of collective struggle and triumph. More importantly, perhaps, they've learned that the programming world isn't locked behind some exclusive door. And just as their favourite glen, rivers, mountains, and woodlands; It's theirs to explore.

References

Desvages, Charlotte, Brittany Blankinship, Umberto Noè, and Pawel Orzechowski. 2026. "Structured Group Work with Assigned Asymmetrical Roles and Switching: Lessons from Pair Programming Across Disciplines." In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Interdisciplinary by Design: The Centre for Data, Culture, & Society Training Programme

Lucia Michielin

Edinburgh Futures Institute, University of Edinburgh
lucia.michielin@ed.ac.uk

Lisa Otty

Edinburgh Futures Institute, University of Edinburgh
Lisa.Otty@ed.ac.uk

Introduction

This chapter discusses the Centre for Data, Culture & Society's (CDCS) training initiative, which aims to integrate computational methods into humanities research. CDCS is committed to fostering methodological innovation and advancing digital research skills within the academic community. It provides targeted support and expertise to both individual researchers and groups, enhancing their capacity to employ digital techniques in the arts, humanities, and social sciences. Importantly, the initiative bridges the gap between technology and traditional research methods, preparing scholars to adeptly incorporate digital tools into their work. The CDCS training programme is distinguished by its interdisciplinary and peer-led educational model, which serves researchers at different stages of their career and with very heterogenous levels of digital literacy. The programme fosters a shared language across disciplines, enabling scholars from non-technical backgrounds to embrace coding-based methods confidently. A natural outcome of this approach is that it enables us to observe how the newly acquired shared language facilitates collaboration and interdisciplinary work, effectively breaking down traditional disciplinary barriers. This aspect is particularly crucial in enabling researchers trained in non-technical disciplines to confidently navigate and employ complex digital tools in their work.

The Centre for Data, Culture & Society (CDCS)

Today's researchers work in a rapidly changing technological environment, tackling complex global challenges: the ability to work across disciplines, scales, and sectors is increasingly important. In this context many researchers are embracing data-led approaches and there is an increase in demand for training in the kinds of applied methods that would traditionally have been taught only in computer science and STEM disciplines through coding-based approaches. Scholars in the arts and humanities, who bring invaluable critical, creative and ethical perspectives to interdisciplinary projects, face specific challenges in acquiring these technical skills: barriers include confidence, grounding assumptions, language and a lack of tailored skills development opportunities. The inherent complexity of computational methodologies, along with unfamiliar technical language and processes, can prove a significant stumbling block to researchers whose time and resource is limited. Empowering such researchers to engage and explore the value of these approaches supports not only more effective collaboration, but also more nuanced, reflective and ground-breaking interdisciplinary projects.

The Centre for Data, Culture & Society (CDCS)[171] at the University of Edinburgh[172] was founded with a mission to inspire and support scholars across the arts, humanities and social sciences to explore, navigate and utilize the potential of data-led and computational methods. To build capacity and support for interdisciplinary digital research projects, and to support the sharing of relevant knowledge, methods and

[171] https://www.cdcs.ed.ac.uk/

[172] Founded in 2019, CDCS is a strategic initiative of the College of Arts, Humanities and Social Sciences at the University of Edinburgh. Since 2021 it has been part of the Edinburgh Futures Institute, one of five innovation hubs established through the Data-Driven Innovation Programme funded by the Edinburgh and South East Scotland City Region Deal. Alongside its training programme, CDCS offers events, technical services and infrastructure for data-led research.

resources between subject areas and schools, it was deliberately positioned outside of academic departments: support from a cross-school academic advisory board ensures that all of the research communities within the College of Arts, Humanities and Social Sciences can feed into planning and prioritization. Through a range of technical services, CDCS offers a space for experimentation and tries to catalyze innovation, fostering a culture of peer-collaboration and knowledge sharing. It provides tailored and flexible support including bespoke technical support and advice, prototyping sandpits, data clinics and opportunities to develop communities of practice. This collaborative ecosystem is further enriched by regular networking events and workshops that provide flexible opportunities for professional development and interdisciplinary engagement. The Centre is staffed by a professional services team that includes a director, training manager, and events coordinator as well as training fellows and research software engineers.

The CDCS Training Programme

The CDCS training programme is a key part of the Centre's offer and has been developed over the last six years to provide participants with practical support in applying computational methods to their projects. It is designed to be inclusive, accessible, and practical, and offers a variety of formats to accommodate diverse learning preferences, ranging from workshops and seminars to intensive boot camps, self-paced learning materials and online courses. 'Digital Method of the Month' sessions, for example, provide beginner-friendly introductions into a specific approach or technique.[173] These hour-long sessions are designed to give researchers a practical sense of what each method involves before committing significant time to it. They create a welcoming, entry-level space for exploring new approaches, with frank discussion of tools, challenges, and the realities of applying methods in practice.

A foundational principle of the training programme is that it is 'by researchers, for researchers' and delivered by a cohort of Training Fellows (TFs), leveraging peer-led instruction to build competencies within a supportive community. The fellowship program recruits doctoral candidates and early-career researchers who use computational methods in their projects.[174] It offers them an opportunity to deliver courses, teach and mentor their peers, thus fostering an environment of reciprocal learning and reinforcing their own understanding of the subject matter. This peer-led structure serves to nurture networks of support and collaboration. A key example is the Centre's Bring Your Own Data (BYOD) sessions, which enable students to bring their work into the learning environment, discuss their processes and problems and troubleshoot together. This allows participants to get hands-on experience and supports learners in translating their new skills to their own research environments, thus reducing barriers to implementation and encouraging ongoing usage of computational methods.

One of the main challenges in teaching computational methods to researchers in the Humanities and Social Sciences is the technological language barrier. Many participants enter workshops without a shared vocabulary for describing what they are trying to achieve through the application of code-based tools. The Centre tackles this by intentionally focusing on building a shared terminology that helps attendees build up their understanding while mapping their research needs against specific computational methods. Instructors provide clear definitions of key terms, introduce foundational structures, and carefully explain the "grammar" and syntax of computational processes. This

[173] For each meeting, we also prepare a Markdown file that provides attendees with starting notes to support their work on a specific method.

[174] The Training Fellows are recruited annually (with the possibility of extending their contract) through an open call to current PhDs and ECRs, and through a standard interview panel. Once appointed, they are offered a Guaranteed Hours contract.
This provides flexibility, assuring them sufficient hours to design and deliver one training course and leaving open the possibility of taking on more work, if they have capacity and interest. How many hours each TF works every year can vary considerably and is dependent on programming, availability, skills, and budget. Once they start their contract, they receive a general induction covering our aims and delivery methods, past training courses and available materials. After that they work with the Training Manager to design the programme.
Each year there is a combination of returning and new Training Fellows, and during the first semester the new TFs normally start as helpers in courses led by more experienced Training Fellows or by running courses where the material has already been consolidated and just needs a simple refresh and update. During this phase, they learn how to provide feedback on material delivered by others, and they familiarise themselves with different methods of delivery. Thereafter, more often in the second semester, they are invited to submit ideas for new training material or more consistent revisions of existing training material. At each stage, they receive and give feedback on materials and delivery methods, both from their peers and from the Training Manager (we use GitHub for our materials as it supports this constant review process). The pedagogical progress of the Training Fellows — from helping in a class led by someone else to producing brand new training material—is usually dependent on their previous experience and capacity, but by the end of their contract, they will all have experience in developing new or extensively revised training material.

approach equips participants with language that they can use to continue learning independently – as we always tell learners, half of the battle is knowing what words to google!

Concepts are made more accessible through metaphors and examples drawn from the humanities and social sciences. For instance, the CDCS Training Manager Lucia Michielin, often introduces the process of building working code chunks by way of a comparison to translating dead languages such as Ancient Greek or Latin. This analogy is grounded in real learning experiences as it relates to her experience as a Classics PhD, trying to find a way to connect unfamiliar computational methods to research tasks she already understood. Training Fellows are encouraged to foreground their own learning experiences and the analogies they have found helpful, which gives trainees insight not only into the method at hand but also into others' learning processes and how they can be adapted to different contexts.

Lessons pay close attention to vocabulary, avoid unnecessary jargon, and focus on core principles, ensuring that no prior computational knowledge is assumed. Participants are encouraged to discuss, consolidate, and actively apply new terminology, reinforcing understanding and enabling them to articulate complex ideas in computational terms. The learning process is framed as analogous to acquiring a new language: mastering basic structures first, then building fluency through repeated practice and application.

Given the international dimension of researchers at the University of Edinburgh, this language barrier can sometimes extend to natural language itself. International students and researchers may struggle to find the right terminology to articulate their needs, particularly when confronted with frustratingly obscure error messages. A helpful strategy is to encourage them to set their coding interface to their native language. This forces them to interpret and translate the error into English when explaining it to the instructor. This mental process of navigating between languages not only clarifies the issue for the instructor but also helps the learner develop a deeper understanding of the underlying computational problem and strengthens their ability to articulate technical challenges in a second language.

By establishing a shared language, the Centre also promotes interdisciplinary collaboration. Researchers from different backgrounds gain confidence in approaching computational problems, bridging gaps between disciplinary cultures. This not only enhances the immediate learning experience but fosters sustained engagement with computational methods: Participants are more likely to translate their new skills into their own work, reducing barriers to implementation, and encouraging ongoing use their learning. As a shared language is developed, it also provides a common framework for exploring questions and problems. A good example of this is the success of the "Data Surgeries" sessions, which provide one-to-one meetings where researchers receive tailored advice on applying digital and data-driven methods to their projects. These hour-long consultations are led by our Training Fellows, who often come from different disciplinary backgrounds than the researchers they are assisting. Fellows help with data processing, troubleshooting, and guide participants toward additional resources or further training.

Open Training Materials

Alongside the Training Fellow-led programme of courses and workshops, the Centre focuses on reinforcing an interdisciplinary and peer-led approach through a range of formats and materials. Since July 2020, the Centre for Data, Culture and Society has been a member of the Programming Historian[175] Institutional Partner Programme, a partnership that reflects our sustained commitment to open, peer-reviewed, and sustainable training resources. In practice, many *Programming Historian* tutorials have been incorporated into CDCS activities: they are used in *Masterclasses* (ad hoc deep dives into innovative computational methods for research) and in *Silent Discos* (asynchronous online workshops; see Cooling et al. (2026) in this volume).

The tutorials also underpin the "further reading" sections of the Centre's structured Training Pathways[176], guiding learners towards high-quality, freely accessible materials that extend beyond the workshops themselves. Covering diverse topics such as data visualisation, statistical analysis, text analysis, sentiment analysis, and working with 3D Data, these Training Pathways are designed to provide structured, accessible guidance through a wide range of digital research methods. They are aimed at beginners and offer a scaffolded approach that helps learners identify which skills to acquire first and how to progress: each pathway highlights essential steps, tools, and concepts, while providing guidance on potential challenges and pointers to areas for further exploration. Rather than presenting methods in isolation, the pathways emphasize the connection between methods and underlying skills, highlighting often overlooked but crucial practices. For example, they make clear that even the most polished data visualisation is

175 https://programminghistorian.org/en/

176 https://www.cdcs.ed.ac.uk/training/training-pathways

meaningless if the underlying dataset has not been rigorously collected and carefully cleaned. By drawing attention to these foundational steps, the pathways reveal the “hidden” work that underpins robust and reliable research outcomes, and, by combining practical instructions with curated “further reading,” they give learners confidence and direction, enabling them to approach computational methods incrementally rather than feeling overwhelmed.

Since its launch, the Centre has produced a substantial body of training materials designed to be open, adaptable, and widely accessible to any discipline. These resources are collaboratively developed by a network of researchers from diverse research backgrounds, ensuring that all materials are proofed and checked to ensure they are jargon-free and speak effectively to a general scholarly audience. All materials are shared openly via GitHub repositories[177] under a CC-BY 4.0 license, ranging from markdown help pages to PowerPoint presentations and notebooks (mostly R or Python based). This open model not only ensures that the resources can be freely accessed and reused but also encourages continuous refinement and improvement based on user feedback and emerging research needs.

Our Impact

Overall, CDCS plays a pivotal role in removing barriers that have traditionally hindered the use of computational methods in various disciplines. This flexible approach has proven successful, with around 600 to 700 researchers per year benefiting from the Centre’s work. By emphasizing interdisciplinarity, peer-support, flexible formats, and explicitly showing learning processes, the Centre is effectively transforming how researchers in the arts, humanities and social sciences engage with digital and computational technologies. This approach not only enhances individual research capacity but also contributes to the broader academic and societal understanding of data, culture, and society, leading to innovative solutions to complex global challenges. As computational methods continue to weave into the fabric of research across disciplines, initiatives like those at CDCS will be indispensable in preparing researchers to navigate and lead in this new digital era.

[177] https://github.com/DCS-training

Appendix 1: Digital Method of the Month (DMM) Session in Detail

DMM sessions are informal meetings to create a safe space to freely discuss the practicalities of learning and implementing a new digital skill in research. Each month a method is selected, and the meeting itself is a 1-hour practical discussion on what it takes to learn and master the method. During the meeting, we try to answer the following questions:

- What this method is really about?
- How much time will it take to get the basics?
- What are the software options available?
- What are the most common pitfalls?
- Where can you find more info on the subject?

A typical schedule is as follows:

- 00:00-00:10 Housekeeping and intro
- 00:10-00:25 Round of Presentation of the attendees (to collect specific interest and hurdles of the attendees)
- 00:25 –00:50 Discussion using a Markdown document
- 00:50-01:00 Discussion on available resources for learning, way forwards and wrapping up

.

.

.

.

Example: Digital Method of the Month on Working with GIS and Geographical Datasets

Learning objectives:

- Become familiar with the main concepts of Geospatial Data and GIS
- Understand the easiest pitfalls new practitioners can run into when starting working with these tools

Materials provided:

- Housekeeping slides with rules of the DMM
- Markdown file with major points of discussion written out with hyperlinks to further resources (all previously run DMM Markdown files are available online178)

Main discussion points:

- Concepts and jargon that practitioners need to familiarize with — e.g. difference between vectors and raster, and practical aspects of working with different data sources, standards of data (like shapefiles, GeoPackages, GeoTiff, etc.), downloading data vs. connecting to OSM and WMS services, etc.
- Time frame for learning and important theoretical topics — e.g. the importance to fully understand how reference systems work and what are the consequences of picking the wrong reference system
- Easy pitfalls for newbies — e.g. propriety vs open source extensions, information tree organization, etc.
- Future possible issues — thorny issues that learners will encounter down the lines once they will move to more intermediate and advanced techniques (e.g. implementation with coding environments, working collaboratively, managing 3D data)
- Main software available, with pros and cons of the main options
- List of helpful websites/blogs on where to ask for help
- Resources to learn GIS, both as self-learner and within the Centre and the University in general

[178] https://github.com/DCS-training/Digital-Method-of-the-Month/tree/main/DMM%20Docs

References

Cooling, Chris, Nick Jayanth, Samantha Alvarez-Madrazo, Leila S. Shafti, Joseph El Gemayel, and Lucia Michielin. 2026. “Ways to Teach Online: Lessons Learned Through Experience.” In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Changing Civil Engineering Students' Mindset Toward Programming

Nguyen Q. Chien

School of Engineering, University of Edinburgh

Introduction

It is high time that students from civil engineering (CivEng) get on with some serious programming. Civil engineering involves creating solutions to real-world problems (Wood 2012). As such, each solution tends to be unique and associated with the particular environmental and societal circumstances pertaining to the local community. It is no surprise that computing had early been adopted in CivEng, with programming classes added to the curriculum, as reported in a US survey (Hoffbeck et al. 2016). After all, CivEng students cannot just sit there and look at their university friends doing all kinds of interesting stuff with code or AI/machine learning. But what are the difficulties behind the scenes? And why do I focus here on such a narrow engineering discipline?

Civil engineers have good logical ability, especially deduction, using and manipulating mathematical expressions, dealing with real dimensions, approximation and optimisation. They have a firm belief in the deterministic: an expected system for this input must yield that output.

Until they start to realise that the computer system (or the compiler, but anyway the students don't care about this) tends to play tricks on them. For example, a code snippet copied from the online tutorial doesn't work, or merging two innocent-looking code chunks throws an error. That's when they realise that they will need a good approach to programming. The class instructor, too, will need to dedicate time to create good code examples as teaching materials — perhaps examples that illustrate some key principles in programming but merge seamlessly with domain knowledge.

This chapter presents some of my experience in tutoring two classes in Thuyloi University (Vietnam), one for final-year undergraduates and one for an MSc class, with use of the Python and Julia programming languages. In both cases, instead of delving into the merits of each language or making any comparison between them, I focus on the practical aspects of solving a typical civil engineering problem — sediment transport. For this problem, we are interested in the amount of sand moving in natural bodies of water (e.g. lakes, rivers) and how the bed elevation of such lakes/rivers changes over time, which has vast applications in real development projects. There is some maths and physics involved, but I will provide simple explanations. Hopefully, by the end of the chapter we can pinpoint some "good practices" for engineering students starting to write their own code. The chapter may also benefit CivEng class instructors who are aiming toward more effective lectures.

Code Implementation

Logic of Calculation

CivEng students generally do not encounter any difficulty in implementing a series of formulae in order, as well as translating a formula to code. For example, in the Python snippet below, the drag coefficient is readily calculated according to the math equation:

Math

$$C_D = \left[\frac{0.4}{1 + \ln(z_0/h)}\right]^2$$

Code in Python

```
CD = (0.4/(1 +
math.log(zo/h)))**2
```

CivEng students are well aware of complex engineering formulas and would be willing to write lengthy statements for mathematical expressions. Furthermore, they have adopted the style of naming variables according to the symbolic representation which is widely agreed in the literature. For example, here the name `CD` is used instead of choosing a more explicit name like `drag_coefficient`.

Even a branching construction would not cause much difficulty, for example to calculate the critical flow velocity using Python. The CivEng student surely would not get confused with units during computation, as they can handily type in conversion factors along with the code:

Math

$$U_{cr} = \begin{cases} 0.19\,(d_{50})^{0.1}\log_{10}\left(\frac{4h}{d_{90}}\right) & \text{for } 0.1 \le d_{50} \le 0.5 \text{ mm;} \\ 8.5\,(d_{50})^{0.6}\log_{10}\left(\frac{4h}{d_{90}}\right) & \text{for } 0.5 \le d_{50} \le 2 \text{ mm;} \end{cases}$$

Code in Python

```
if 0.1E-3 <= d50 <= 0.5E-3: # mm
  Ucr = 0.19*(d50**0.1)*math.log10(4*h/d90)
elif 0.5E-3 < d50 <= 2E-3: # mm
  Ucr = 8.5*(d50**0.6)*math.log10(4*h/d90)
else:
  print('Warning: d50 out of range to apply
formula for Ucr.')
```

Dimensions and Units In Civil Engineering

A particular strength of the CivEng student is having keen eyes on physical quantities. The branching statement presented above shows that the student was aware of units and applied corresponding coefficients. The ability to tell meaning from numbers allows them to check intermediate results and correct potential errors during computation. Another advantage is that by adhering to the rules of dimensions in physics, students naturally familiarise themselves with the type system and therefore will not have much difficulty with languages like C#, Java, or Scala, or writing type annotations in their Python code.

Matrix Manipulation

In many circumstances, the CivEng student has to deal with matrices, such as when solving partial differential equations (PDEs). One of such typical equations is the mass balance equation, where the (time) change in "stock" is related to the spatial changes (gradients) of "flows". It is not too difficult for the CivEng student to understand the finite difference technique and implement a naive version to solve PDEs. As an example, in the problem of sediment transport in rivers or seas, we consider solving the equation for z_b (the underwater bed elevation), which changes in time, given the two fluxes of sediment, represented as two components (q_x and q_y) in the two horizontal directions, x and y. It is helpful to visualise the river bed or seabed as a "tiled" surface where each cell has its own elevation z_b. See the diagram (Figure 39.1) for a simplified representation of the quantities involved.

It should be noted here that the CivEng student is equipped with numerical analysis skills and can approximate the differential in terms of finite differences.

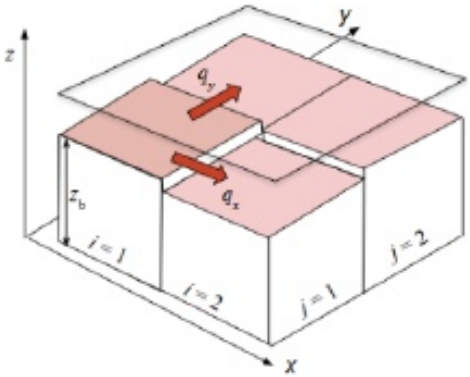


Figure 39.1: Sediment transport

Differential equation

$$\frac{\partial z_b}{\partial t} = -\frac{1}{1-n}\left(\frac{\partial q_x}{\partial x} + \frac{\partial q_y}{\partial y}\right)$$

where z_b, q_x, and q_y are functions of x and y. Therefore, we will also make $\frac{\partial z_b}{\partial t}$ a function of x and y.

Numerical approximation

$$\left(\frac{\partial q_x}{\partial x}\right) = \frac{q_{i,j} - q_{i-1,j}}{dx}$$

Here the numerical approximation for one flux gradient term is illustrated.

The naive approach is to use a nested list to emulate a 2-D array, which doesn't seem very efficient. But that's enough for the CivEng who wants to get the job done. This way, the code is more transparent when using an index system with $(i \pm 1)$. Notably, there is similarity in operators and operands between Python and Julia (thus I will place ellipses in the Julia code). The differences are in for-loop syntax and the 1-based indexing system in Julia as opposed to Python's 0-based. A notebook implementation[179] is provided online.

Code in Python

```
for i in range(1,nx):
  for j in range(1,ny):
    dqxdx[i][j] = (qx[i][j]-qx[i-1][j])/dx
    dqydy[i][j] = (qy[i][j]-qy[i][j-1])/dy
    dzbdt[i][j] = -(1/(1-n)) * (dqxdx[i][j] +
dqydy[i][j])
```

Code in Julia

```
for i = 1:nx
  for j = 1:ny
    dqxdx[i,j] = ...
    dqydy[i,j] = ...
    dzbdt[i,j] = ...
  end
end
```

[179] https://nbviewer.org/url/coastal-study.uk/sed.trans/SedTrans.ipynb

The use of matrix operations (e.g. using NumPy[180] arrays in Python) is faster compared to the "naive" version above, usually quoted as 10 times (Langtangen 2006). On the downside, the matrix notation obscures the indices `i` and `j` commonly found in numerical methods (see Figure 39.2, where part of a stencil[181] is shown. In this part, where arrows represent subtraction, we are finding "local" difference in the same `qx` array). When using matrix manipulation, the bounds must be specified correctly in all matrix subscripts (see Figure 39.3 and the code below). So, there should be a consideration in choosing either approach.

Looping through index

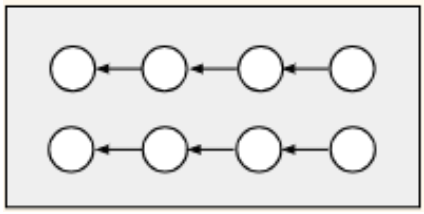


Figure 39.2: Lattice 1

Code in Python

```
dqxdx[i][j] = (qx[i][j]-qx[i-1][j])/dx
```

Code in Julia/Python

```
dqxdx[i,j] = (qx[i,j]-qx[i-1,j])/dx
```

Matrix-based computation

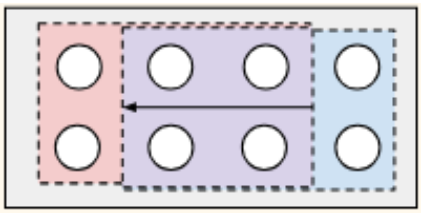


Figure 39.3: Lattice 2

Code in Python

```
dqxdx[0:-1,:] = (qx[1:,:] - qx[0:-1,:])/dx
```

Code in Julia

```
dqxdx[1:nx-1,:] = (qx[2:nx,:]-qx[1:nx-1,:])/dx
```

Another source of confusion is due to the concept of "views" in NumPy[182], for which the submatrix, e.g. `dqxdx[0:-1,:]` in the code snippet above, is a *view* and not quite a *copy* of the array `dqxdx`. In essence, NumPy "views" will behave in unexpected ways. Sure, this can be explained to the students, but every such issue can add up and discourage the learners. Therefore, an "agreement" might be made beforehand, that the matrix `qx` here is calculated once and cannot be modified thereafter. Otherwise, you must switch to another programming language, e.g. MATLAB, although this comes with its own issues.

[180] https://numpy.org/

[181] A stencil is a diagram showing a group of computational nodes where numerical approximation takes place.

[182] https://numpy.org/doc/stable/user/basics.copies.html

Code Reuse

To CivEng students, reusing code is actually more difficult than writing new code! While the students can implement the code quite well in terms of logic and calculation (see the previous section), they somehow feel embarrassed when having to perform the calculation again with a slightly different input data, or in the CivEng's language, another "scenario". Given the intuitive layout of code blocks in a notebook, the students tend to just copy-and-paste code and make minor edits, which leads to code with excessive repetition. Other related "gotchas" are discussed in other chapters of this book (El Gemayel, Budiarto, and Bell 2026; Skipsey et al. 2026).

During my tutorials in the past, students were exposed to a limited amount of pre-written code in the form of Jupyter notebooks only. By reading these Jupyter notebooks, the students were able to pick out relevant code cells and run them. They would know which chunks of code should be executed and in what order, so that no errors emerge showing undefined variables. But when students copy chunks of code from the sample file to their *own* notebook, renaming variables is often required.

Another issue is lexical scoping. Being more acquainted to "physical" objects rather than information "pieces", CivEng students would insist on using different names for variables appearing in both the inner and outer scopes despite the fact that these names would not conflict. This is reasonable for array data types, as in the case below, where the CivEng student creates a `zb_init` variable to store the initial state of the system (and the use of `copy()` to avoid the potential issues with using matrix views mentioned above). However, the situation is different for single numbers (scalars). Consider a variable for the characteristic "near-bed" flow velocity, `Uf`, which is created outside and being updated inside the function body, without affecting its original value.

```
zb = np.zeros((nx,ny))
zb_init = np.copy(zb)  # initial value
dt = 2.0  # const param
S = 1E-4  # const param
zs = 1.0  # const state variable
Uf = np.sqrt(9.81 * zs * S)


def evolve():
  for i in range(1,nx):
    for j in range(1,ny):
      zb[i][j] += dzbdt[i][j] * dt

  Uf = np.sqrt(9.81 * (zs - zb.mean()) * S)


for _ in range(nt):
  evolve()


print('Initial zb:', zb_init)
print('Initial Uf:', Uf)
```

[183] https://nbviewer.org/url/coastal-study.uk/sed.trans/SedTrans.ipynb

Code Development

Mental Orientation

An important aspect of moving from a CivEng to a programmer is to re-imagine the conceptual model for the system. Compare:

	CivEng	CompSci
Para-digm	imperative, procedural	object-oriented
Object type	concrete, physically meaningful	abstract, blueprint to generate instances
Action	math and logic operations applied to parameters of the object	objects' methods invoked on instances

Admittedly, it is not easy to persuade the student to adopt an OOP (Object Oriented Programming) style of programming to benefit future use. While for certain disciplines, e.g. Chemistry, the notion of class is inherent (e.g. class of elements), the CivEng is more inclined to think about real-world entities: a particular building, canal, etc. to be modelled.

Unfortunately, the common situation for CivEng is that many handbooks and guidelines adopt the procedural approach which can be summarised as follows:

1. Check the set of inputs against some predefined criteria.
2. Follow an established method step-by-step (such as the one in the SedTrans notebook[183] mentioned earlier) to arrive at some output (such as the vertical change in seabed level, `dzbdt`).
3. Evaluate the result, `dzbdt` — is it in a reasonable range? Is it realistic, and how does it compare to similar

studies in the past (“literature”)? Furthermore, is the computational routine *stable*, i.e. the numerical error does not grow up unbounded, through many iterations.

Apparently, most of the computational workload here lends itself to imperative programming. We could think of OOP aspects in organising a class of problems presented in step (1), or a class of test cases in step (3), but that is all. CivEng still relies on researchers in physics whose perspectives clearly align to the model: data in → transformation → data out.

Generalisation

CivEng students understand the use of functions and are adept in translating formulation of various calculation routines into functions. So that should be enough for generalising code, shouldn’t it? During a practice session, after having students estimate the derivative $f'(x_0)$ for the specific functions $f(x) = x^2$ and $f(x) = \cos(x)$ with finite differences, I required them to generalise their `deriv_*` function to accept any mathematical function. Well, how can we do that? — they doubted, until I showed that we can just pass the function under investigation as one of the inputs of the function `deriv` (see the Julia code below) — a first step toward functional programming.

Student’s version

```
deriv_sq = function(o, h)
  ((o + h)^2 - (o - h)^2) / 2h
end

deriv_cos = function(o, h)
  (cos(o + h) - cos(o - h)) / 2h
end

deriv_sq(0, 1E-3)
deriv_cos(0, 1E-3)
```

Recommended version

```
# A generic central-difference approximation
deriv = function(f, o, h)
  (f(o + h) - f(o - h)) / 2h
end

deriv(x -> x^2, 0, 1E-3)
deriv(x -> cos(x), 0, 1E-3)
```

Tooling

The students are happy with Jupyter notebooks during class. Syntax highlighting, code auto-completion, and embedded plot outputs are features that are highly appreciated. Nevertheless, a dedicated environment like VS Code is needed for students who want to develop larger programs with debugging and version control (e.g. using git). In another chapter of this book, El Gemayel, Budiarto, and Bell (2026) mention the use of a Python debugger (pdb) in the Jupyter Notebook environment, yet the more recent Jupyter Lab features a more convenient visual debugger so that students can track the changes of variables during the flow of the program.

Recently, there has been an increasing tendency to use AI in coding. While AI tools are a great boon for syntax checks and documentation, it would be detrimental to have AI chatbots generate the code skeleton and logic, as this part requires the ingenuity of civil engineers who possess the domain knowledge.

Good Practices

This list is non-exhaustive yet would be useful for CivEng students starting to develop their code:

- Variable names should follow the widely adopted **notations** in civil engineering. When possible, use the Unicode characters for Greek letters, e.g., using `α` instead of `alpha` (subject to the language's feature — this is favourable for Julia with keyboard shortcuts for such characters/symbols).
- Extensive and meaningful **documentation** is needed since there are many empirical formulae used in CivEng and the choice of a formula should be justified based on the context.
- Whenever the names are cluttered, consider using **namespaces** (knowledge of the engineering specialisation is required.)
- If facilitated by the language (in the case of Julia), prefer using **loops with index variables** (`[i±1]`) rather than implicit array position index (`[:]`).
- Generalise code with **functions**, so that you can reuse it later. This is particularly useful in CivEng, which heavily relies on various recipes. Each recipe can be implemented as a function.
- Using **advanced features** in Jupyter Lab or an IDE to organise and debug code (first step), and then aim toward collaboration, version control and testing.

Final Thoughts

In summary, although there is not much material available for CivEng students' code development, there are patterns they may adopt to boost their performance at work. In this chapter, I present some such common patterns. Due to the varied nature of the disciplines that a CivEng may be involved in, it by no means presents a single context that they can build programs upon. Rather, I present a case of water/hydraulic engineering and sediment transport, showcasing field variables, spatial derivatives and implementing a finite difference numerical solution.

Given the ubiquity of materials and tools to learn and practice programming nowadays, CivEng students can leverage their skills to solve specific problems in their field. However, unlike data science where standardised software libraries have been developed, CivEng students may often find themselves coding from the ground up. In such situations, being able to design their program is crucial; and this can be accomplished through extensive practice with good programming patterns and continually learning to shift the mindset from "computing with numbers" to dealing with data organised on your computer. This is not easy, though, and I would refer to Mindstorms (Papert 1980), where the author used Turtle's geometry to dispel the "mathophobia" among schoolchildren.

These days, with programming a mandatory part of CivEng students' skill set, it is best for them to appreciate programming as a way to represent their own workflow and reach a solution to their engineering problems, which can then be evaluated and improved.

References

El Gemayel, Joseph, Arif Budiarto, and William Bell. 2026. "Notebook for Novices? Pros and Cons of Jupyter." In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Hoffbeck, Joseph P, Heather E Dillon, Robert J Albright, Wayne Lu, and Timothy A Doughty. 2016. "Teaching Programming in the Context of Solving Engineering Problems." In *2016 IEEE Frontiers in Education Conference (FIE)*, 1–7. IEEE; IEEE. https://doi.org/10.1109/FIE.2016.7757617.

Langtangen, Hans Petter. 2006. *Python Scripting for Computational Science*. Springer. https://doi.org/10.1007/978-3-540-73916-6.

Papert, Seymour A. 1980. *Mindstorms: Children, Computers, and Powerful Ideas*. Basic books. https://doi.org/10.1007/978-3-0348-5357-6.

Skipsey, Sam, Gordon Stewart, Jeremy Singer, and David Cutting. 2026. "Bridging Languages: Teaching c to Python Novices." In *Teaching Programming Across Disciplines*. Edinburgh Diamond.

Wood, David Muir. 2012. *Civil Engineering: A Very Short Introduction*. Vol. 331. Oxford University Press. https://doi.org/10.1093/actrade/9780199578634.001.0001.

Lost in Translation: Complexities and Good Practices of Translating Coding-Based Teaching Materials Across Multiple Human Languages

Olexandr Konovalov

School of Computer Science, University of St Andrews
obk1@st-andrews.ac.uk

Lucia Michielin

Edinburgh Futures Institute, University of Edinburgh
lucia.michielin@ed.ac.uk

Yanina Bellini Saibene

rOpenSc; RLadies+; Universidad Austral
yabellini@gmail.com

Introduction

Translating coding-based teaching material may look like a straightforward practice, especially if you are translating from English to your native language. However, this process is often undertaken in contexts where learners are already navigating coding concepts through a second language, which adds an additional layer of difficulty. If you are directly involved in teaching such students, or participate in international educational projects like The Carpentries[184] or Programming Historian[185], you may have considered translating teaching materials to make them more accessible.

Based on our experience in translations and localisation projects, we decided to write this short contribution to help others navigate similar situations and avoid common pitfalls.

This contribution is a collection of personal reflections and good enough practices we learned through the years, with the caveat that any translating project will have its own peculiarities and needs. While many of our reflections stem from medium to larger, collaborative efforts, some of these practices may also be adapted to smaller-scale projects, although this is not the primary focus of this chapter.

General Considerations

Before moving into the practical side of this contribution (organised around the different project stages: designing, deploying, and maintaining), we wanted to outline a few general considerations.

As mentioned earlier, it is often the case that translation occurs from English into another language that is the translator's native language (at least this was our shared experience), reflecting the dominant role of English in coding and academia. While this chapter focuses primarily on that direction, other forms of translation may be required in different contexts.

Translating from another language into English follows essentially the same processes and can rely on the same tooling as translation from English into other languages. It also requires establishing shared agreements within the community, just as in any other translation effort. For example, the rOpenSci translation guidelines[186] include language-specific chapters that document community decisions on how to approach translation into each language, highlighting that these agreements are integral to the workflow regardless of the direction of translation.

Moving from a language into one's native language may create a misleading sense of reassurance: if one was able to design and teach material in a language that is not their primary one, it might seem reasonable to assume that doing so in their mother tongue would be considerably easier. This is not always the case.

First of all, if you have studied a subject in a certain language (e.g. English), it is harder to then convey the core topic in a different one. Few situations are more frustrating than knowing how to refer to a concept in a second language but being unable to recall the equivalent term in one's native language, and this experience is likely to recur frequently throughout the process.

Because English has been the vernacular language for programming languages, their official documentation and most of the coding-based teaching material available on the internet, the word you are searching for may not actually have a clear translation into the other language. This brings us to another fundamental issue: how much should you actually translate? Where do you stop? The aim is to create material that is understandable for your audience, while also providing them

[184] https://carpentries.org/
[185] https://programminghistorian.org/
[186] https://translationguide.ropensci.org/

with enough background to refine and progress their knowledge beyond your course.

If you fully translate everything and introduce many neologisms, you risk preventing learners from efficiently searching for further information online. On the other hand, if you do not translate and explain key concepts and jargon sufficiently, students may struggle to fully comprehend the content. It is a fine line to navigate, made even more complex when dealing with non-Latin alphabet languages, where transliteration also becomes a concern.

No single solution fits all, and much will depend on the language in question. In collaborative projects, community discussion is often central to these decisions, making the process more collective and sustainable over time. As a rule of thumb, it is helpful to translate terms that can become established jargon in the target language, while leaving others in English. This can be supported through "double-barrelling" keywords, where both the original English term and a more accessible translation are retained.

More generally, any translation project benefits from two key approaches: a collaborative approach, involving people with diverse linguistic and subject expertise, and an iterative approach that allows materials to improve over time. The more the material is used, the more it can be refined. The wider uptake of the material is also a good indicator of a successful translation project, because the main aim of any translation programme is to reach a wider audience and increase overall literacy.

Having discussed these general approaches and issues, we now move into more targeted advice based on the different stages of the project. Although it may seem more applicable to a large and medium-sized translation workflows, you may find it useful to consider even when starting a small personal project, since it may be helpful to set up things in a scalable and reproducible manner from the start (in the same way like setting up a Git repository for a personal task).

Before Starting: The Design Stage

Approaching a translation project, it's important to have a clear idea about its organisation. You are going to face a substantial task. Choosing the wrong tools and process may cause substantial delays later or impose very restrictive limitations.

Internationalisation vs. Localisation

A common view is to separate internationalisation (commonly denoted as "i18n") and localisation ("l10n") steps[187]. Internationalisation is focussed on creating multi-language versions of the product (e.g. book, website, software, etc.), while localisation is usually a separate step which happens later.

In the context of training materials, for example, translation would use the same examples and datasets, as in the original version; an exception to this general rule is the usage of specific units of measure, currency and date format that may not make sense in a translated version. A localised version of the training materials would use variables names, examples and datasets that are more relevant to the language target audience. Separating these two stages makes the first one more achievable.

You also need to decide whether there is a real need in the localisation of examples balancing on student understanding and engagement and maintenance of the translation — if the English source is frequently updated, it will make it harder to keep the languages synchronised. Focusing on internationalisation (meaning translating the main text only) is often a more realistic goal than localisation (which includes translating and adapting more than the main text). Working through this process can help clarify your next steps.

Technical Tools

Next, you need to consider the technical infrastructure for your translation project. There are some aspects to keep in mind for ad hoc approaches without using any of the specialised platforms.

Several translation projects have used tools and roles that are already well known in the world of software development: the project takes place on GitHub or a similar version control platform, organized by GitHub projects where you can track the status of each task and who is responsible for it, where an initial machine translation is performed and presented as a Pull Request, which is then reviewed by usually 2 different people to correct common errors made by machine translators and ensure consistency in the translation in accordance with the agreements of the team. This flow replicates other processes with familiar roles (editors, reviewers) and tools (Markdown, version control), enabling more people to use it more quickly. In addition, those who are unfamiliar with the tool can be trained, which has the added benefit of teaching them a tool that will be very useful beyond the translation project.

A straightforward idea might be to use, for example, a web-based editor such as Google Docs, or, if the source is in Markdown format, a collaborative editor like HackMD, or, if you want to translate a website, just edit raw HTML files. Indeed, this seems to be a viable approach — the first two allow real-time collaboration and leaving comments; the latter two can be used with version control. Not that it is impossible — but it needs to be well thought through.

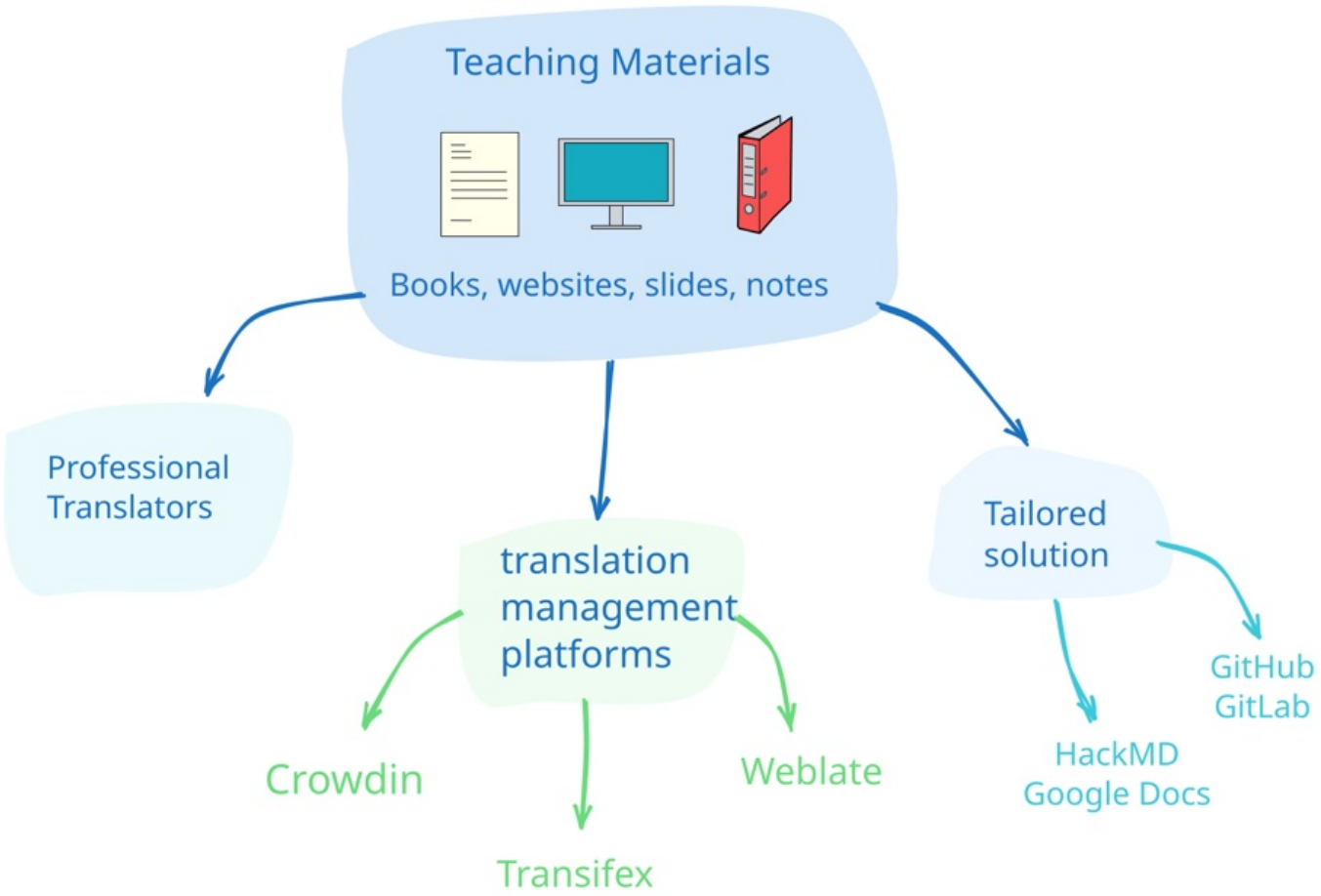
A potential problem is that it does not scale well, especially if the source is not static, and it is necessary to periodically update the translation to keep it in sync with the original. Another challenge is to maintain consistency of translation —

[187] See Wikipedia - Internationalization and localization

when multiple translators are working on different parts of the document, they may translate the same terminology in different ways. Finally, you want to track the progress and easily see which parts of the translated document are already finalised and which need discussion. All these tasks are typical for collaborative translation projects, and are helped by specialised purpose-built systems. Thus, from such a system you can expect a customisable workflow in which each fragment of text can be independently translated, proofread,

updated and approved, possibly by different persons involved at each step. In addition, there should be some semi-automated procedures (so you may have additional quality control steps) to export translations from the platform (i.e. to make them live on a website) and to pull new versions from the original source, indicating which portions of the text have been updated and need to be translated again. The latter operation may be assisted by suggested translations, based on the previous versions — this is what is called “Translation memory”, often abbreviated as TM.

Figure 40.1: Different approaches to translating teaching materials: from professional services to platforms and customized workflows.



In the following table, we describe several existing platforms for collaborative distributed translation: Crowdin[188], Transifex[189], and Weblate[190]. It is not feasible to make a single recommendation because so much depends on the specificity of your project, but we tried to collect as much information as possible in order to make your choice easier.

Table 1: Comparing Platforms for Collaborative Translation

Feature:	**Platform:**		
	Crowdin	**Transifex**	**Weblate**
Type	Cloud, SaaS	Cloud, SaaS	Self-hosted, SaaS
Open source	No	No	Yes
Key integrations	GitHub, GitLab, Bitbucket, Figma, Slack, Jira, API	GitHub, GitLab, Bitbucket, API, Slack, Jira	GitHub, GitLab, Bitbucket, API, REST hooks, custom scripts
Supported content formats	60+ (PO, XLIFF, JSON, YAML, etc.)	50+ (PO, XLIFF, JSON, YAML, etc.)	60+ (PO, XLIFF, JSON, YAML, etc.)
Collaboration	Web UI, comments, tasks, roles	Web UI, comments, tasks, roles	Web UI, comments, review queue
Glossary & Translation Memory	Yes (Advanced)	Yes (Advanced)	Yes (Basic, extensible)
Machine translation	Google, DeepL, Microsoft, custom	Google, DeepL, Microsoft	Google, DeepL, Microsoft, others
Pricing	Subscription, custom plans. Limited free tier (open source projects: free)	Subscription, custom plans. Limited free tier (open source projects: free)	Free (self-hosted, free), SaaS plans (free tier for open source projects)
API/CLI	REST API, CLI	REST API, CLI	REST API, CLI
Mobile SDK	Yes (iOS/Android)	Yes (iOS/Android)	No
Quality checks	Yes (customisable)	Yes (customisable)	Yes (customisable, scriptable)
Workflow Automation	Advanced	Advanced	Highly customisable
Community Projects	Supported (OSS discounts)	Supported (OSS discounts)	Native support
Used by	The Carpentries, The Turing Way	Localization Lab, Open edX	The R-Dev Group

[188] https://crowdin.com/ | [189] https://www.transifex.com/ | [190] https://weblate.org

The Implementation Stage: The Translation Workflow

When it comes to the deployment stage, much will depend on the specific tools and setup you choose to use. However, we have gathered a series of good practices and common pitfalls that should help streamline the deployment phase, whatever your workflow. The following tips are designed to support a smoother, more efficient translation workflow as you bring your teaching materials into multiple languages.

Peer Review

Whenever the translations are made in a specialised platform like one from the table above, submitted via GitHub or GitLab pull requests, or contributed in some other way, it is crucial to have them reviewed. This helps to ensure consistency of translation across the project and to propagate the knowledge within the translation team. Make sure that you provide feedback to the translators, instead of quietly correcting their errors. Timely feedback is especially important when new translators join the team: delayed feedback may result in them repeatedly making the same errors, causing more work to fix them in the future.

Reviewing collaboratively-made translations imposes the need for an approval stage (for example, a button in the web interface). With some reasonable exceptions (e.g. a code example that needs no translation), we recommend following the rule that one should never approve one's own translations or corrections. Instead, consider a ping-pong-like exchange between the translators. When one of them makes a pass over a section of a document, the other approves it or makes corrections and returns it for approval or revision by the original translator. The tools should help you organise such a workflow in a distributed setting.

For languages spoken in multiple countries, revisions should ideally involve reviewers from different regions. For example, in Spanish, it is beneficial to include reviewers from several countries to ensure that the chosen words and expressions are widely understood. Don't underestimate that asynchronous communication will take more time. Sometimes getting together in an online meeting helps to traverse and finalise the remaining fragments of the text much faster and get the next portion of the document ready for publication.

Onboarding New Translators

Onboarding new translators is a regular task in community-driven projects. It is important to prepare guidance and provide training for new contributors. Here, there are a few things that need to be explained:

- **Is it important for your project to provide the exact word-by-word translation, or can one rephrase the text freely to convey the meaning?** When translating a coding-based teaching material, it might be sensible to use a completely different phrase or definition that will be more understandable to the target audience. Doing this, one should find the right balance — diverting too much from the original may lead to inconsistencies with other parts of the material, and may be harder to maintain in the future if the original text changes.
- **Are there any parts of the text that should not be translated?** The source may have a specific format and may include parts used to build the resulting document or website, and these should remain intact. New translators might be unfamiliar with this format and accidentally translate too much. Prevent unnecessary effort by explaining things like specific markup constructions that should not be translated, hyperlinks, cross-references, etc. Explain also whether it is important to preserve formatting and try to approximate line breaks.
- **How to communicate efficiently?** For example, how to reach a user within the collaboration platform (e.g. referring to their username), and how to set up your notifications to avoid missing them. Setting up communication channels efficiently would allow to keep the discussions closed to the translations (e.g. as comments on the pull request or in the translation management system). In addition, consider having a live channel for interactive discussions and technical support (e.g. Zulip or Slack).
- **Which parts of the document are prioritised for translation?** If someone joins in the middle of the project, are there any parts already translated which are recommended to read first? Also, consider a workflow which allows publishing partial translation, for example, the first few sections of the teaching materials. This may allow you to run pilot trainings, and may attract more translators to your project.
- **How to submit improvements to already-made, and possibly even approved translations?**

Glossary

Finally, if the glossary is not a part of the translated document, we recommend establishing it. It could at least be used by the translators; even better, you might consider adding it as an appendix to the translated version. Another useful tip is to add English terms in parentheses when a notion is introduced in the text for the first time.

The Long-Term Maintenance: General Good Practices

While the creation and deployment of multilingual teaching materials are critical steps, it is equally important to recognise that the process does not end there. Digital and web-based resources are inherently dynamic — they need periodic updates as the software changes, educational needs evolve, or the content becomes outdated. Over time, links may break, interfaces may change, and new technologies might require updates to ensure continued accessibility and functionality.

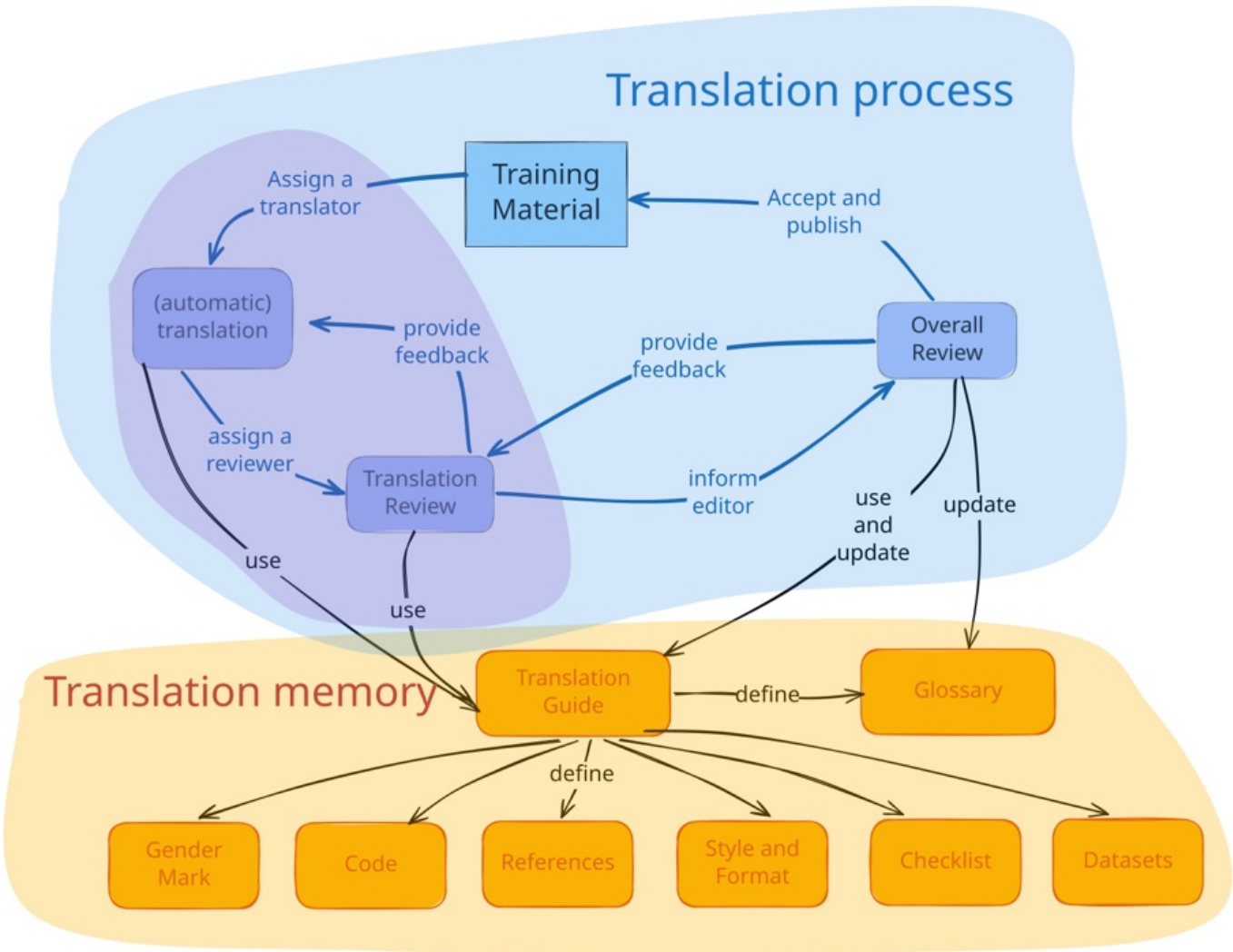
Long-term maintenance, therefore, must be part of your project planning from the very beginning. This means allocating both financial resources and sufficient staff time for ongoing review, updating, and technical support. Consider who will be responsible for regularly checking the content, responding to user feedback, and implementing updates across all language versions. Budget not only for immediate project costs, but for the recurrent, perhaps less visible, work required to keep your materials accurate and effective.

A proactive approach to maintenance prevents issues from accumulating unnoticed, minimises disruption to users, and helps ensure that your investment in high-quality, inclusive teaching resources continues to deliver value over time. With this in mind, we have collected some general good practice advice to help establish sustainable maintenance practices for your multilingual materials.

Use version control! If your workflow is based on contributing translations via GitHub or GitLab pull requests, you are already there. This gives you reviewer functionality and access to other tools from their ecosystem, such as publishing static websites or uploading releases on Zenodo. If you use a specialised translation management platform, the best approach is to use version control anyway. The input and output text may be stored in the

outside the platform, and the platform's integration with the repository hosting service would allow a semi-automated import of the original version and exporting the translation. Using the external repository would reduce the dependency on any particular platform. Remember to keep a record of who worked on the project as translator and reviewer to give them credit — this information will be automatically recorded if you are using GitHub or GitLab, but otherwise you may need to take note of it manually.

Figure 40.2: Translation workflow for training materials, showing how translation, review, and feedback interact with supporting resources like translation guides and glossaries, and style conventions.



It may also happen that while you are working on the translation, a new version of the original text is released. Importing updates into your project may set you back. It might be useful to think of versions of the documents, first publish a full translation of some version, and then import and work on the new one, instead of trying to sync two moving targets, and at no single point having a full translation. This, however, very much depends on the particular project. If it is more important that the document is up-to-date than fully translated, using the specialised platform, you may set up a workflow that renders a document with translated and approved fragments in the target language, and the rest in the original.

Finally, translating code-based teaching materials is a challenge: one should be fluent in the terminology and subtleties in both languages and understand the technical content. Welcoming contributors who do not necessarily possess all three skills, but just one, and are willing to learn through collaboration and feedback, would make your community more inclusive and diverse.

Final Remarks

Language shapes who can access knowledge, participate, and contribute in communities. When most resources are available only in English, a large portion of the global community faces structural barriers that cannot be overcome by individual effort alone. Addressing these barriers is essential not only for being fair but also for the sustainability and global relevance of scientific and open-source initiatives and projects.

We wrote this text based on the experience and/or recommendations from The Carpentries[191] and the Ukrainian Carpentries Subcommunity[192], The Turing Way[193], "Teaching Tech Together" (Enseñar tecnología en comunidad)[194], rOpenSci Multilingual Publishing Project[195], and "R for Data Science" (R Para Ciencia de Datos)[196].

If you wish to explore more hands-on examples of translation projects, you can have a look at the following:

- Multilingual Data Science[197] is a blog by Yanina Bellini Saibene, where she explores her own experience in creating multilingual data science teaching material.
- The documentation for The Turing Way translation workflow[198].
- The documentation for the internationalisation of The Carpentries lessons[199].
- An example of guidance for translators[200] from the Ukrainian Carpentries subcommunity.
- A multi-lingual glossary[201] for computing and data science terms with a GitHub-based contributing workflow.
- The rOpenSci Localisation and Translation Guidelines[202] explain why and how they localize and translate their resources.
- The guidelines for translations of Programming Historian material[203].

Through translation initiatives, we can help lower these barriers by expanding access to knowledge and participation. By creating technical and social infrastructure for multilingual resources and recognising translation as a meaningful form of contribution, we can foster more inclusive and diverse communities and enable more people to engage, contribute, and shape open source projects.

[191] https://carpentries.org/
[192] https://ukrainian-carpentries.github.io/
[193] https://book.the-turing-way.org/
[194] https://teachtogether.tech/es/
[195] https://ropensci.org/multilingual-publishing/
[196] https://es.r4ds.hadley.nz/
[197] https://datasciencebydesign.org/blog/multilingual-data-science
[198] https://github.com/TWTranslation/translation_workflow
[199] https://github.com/joelnitta/carp-i18n-guide
[200] https://github.com/ukrainian-carpentries/translation-guide
[201] https://github.com/carpentries/glosario
[202] https://translationguide.ropensci.org/
[203] https://programminghistorian.org/en/translator-guidelines

Closing Remarks for This Book

Brittany Blankinship

Edinburgh Medical School, University of Edinburgh
b.blankinship@ed.ac.uk

Franziska McManus

Edinburgh Medical School, University of Edinburgh
z.mcmanus@ed.ac.uk

Pawel Orzechowski

Edinburgh Medical School, University of Edinburgh
pawel.orzechowski@ed.ac.uk

You have made it to the end! Well, the end for now...cue the dramatic music <dun dun dun>. Much like our community, which is open and ever growing, this book is a living document. There will be a second version at some point, which expands and fills gaps in the current version. If you have read this far, you may want to join us and contribute!

We hope you have enjoyed reading the diverse range of experiences and perspectives provided by our authors. While the named editors have edited the book and wrangled it into its current format, this book has truly been a grassroots community effort. Our 75 authors have all taken time out of their lives, unpaid, to work together and share their experiences. Our editors have learned new programming languages and tools (Quarto, Lua, Visual Basic), new workflows (github forks), project management and editing skills in order to make this endeavor a success and, in doing so, we have asked our authors to engage with new software and workflows (hands up if you created your first fork on this project). If you were to look up “passion project” in the dictionary, this book would be there.

Get in touch by emailing us at pairprogramming@ed.ac.uk with any questions, suggestions, or feedback. All teachers will recognize the importance of feedback - If something you have read in the book has impacted you or your teaching practice, we would love to hear from you!

Authors' Biographies

Book Editors & Chapter Authors

Brittany Blankinship (she/her)

Lecturer in Data Science, Edinburgh Medical School, University of Edinburgh



Prior to entering the Data Science and Health Tech landscape, my academic background was in Linguistics, History, Psychology, and Neuroscience. In fact, I did not encounter programming until my Master's Degree. This was a truly transformative experience that altered the trajectory of my career. Programming unlocked statistics and so much more - and I have never looked back! Day to day, I am the Programme Director for an online, part-time MSc in Data Science for Health and Social Care. I am committed to empowering students by making programming and data science accessible and enjoyable to learn.

As an author, I bring my diverse teaching experience to the book, which spans learning modality (in-person and online), level of study (undergraduate, postgraduate, and CPD), and topics such as statistics, machine learning, data science, and research methods using R and Python. My interdisciplinary background was the foundation of my interest in co-developing the Pair Programming Community that has led to this book. On paper, I am the Principal Investigator for this book project, but it has truly been a group effort with the wonderful community we have built and our outstanding co-editor team.

LinkedIn: https://www.linkedin.com/in/brittany-blankinship/

ORCID: https://orcid.org/0000-0002-1405-7396

Pawel Orzechowski (he/him)

Lecturer, Edinburgh Medical School, University of Edinburgh



I'm a programmer who fell in love with teaching. I believe that coding is a social and creative activity – it can open up our heart and mind so that we communicate better with each other and computers. For me, coding happens in your head and on paper, as you discuss it with someone... typing it up is just something you do at the end. I've taught programming for over a decade in coding bootcamps and universities (in business, social science and medicine contexts). I've contributed time, technical skulls, words and illustrations towards this unique book. I believe in authentic learning and creativity (without LLM reliance) so every letter you read was typed by a human.

LinkedIn: https://www.linkedin.com/in/dr-pawel-orzechowski/

Personal website: https://www.codestorytelling.com/

Charlotte Desvages (she/her)

Lecturer, School of Mathematics, University of Edinburgh



I am a Lecturer in mathematical computing, teaching Python programming skills, numerical methods, and modelling for mathematics students. I first learned programming for my Physics degree, and although initially it didn't click for me at all, I enjoyed it much more when we got to building physics simulations. Decrypting and implementing algorithms to simulate planetary systems or particle dynamics felt incredibly rewarding, and like a whole new world was opening. I was no longer limited to deriving results from oversimplified models on paper; I could build simulations for much more complex and interesting systems. Later on, I specialised in computational acoustics, building virtual musical instruments based on physics simulations. These experiences continue to guide my teaching practice, which I've further developed around peer learning for programming (via pair programming, code review, live coding); the development of computational thinking linked to mathematical thinking; and accessible and inclusive learning.

Work website: https://maths.ed.ac.uk/people/a-z?person=654

Github: https://github.com/chdesvages

Kasia Banas (she/her)

Senior Lecturer, Edinburgh Medical School, University of Edinburgh



My background is in social psychology, and it wasn't until after my PhD that I discovered my love for programming. But, once I was introduced to R, I fell in love and never looked back. Now I teach introductory data science skills to medics and biomedical sciences students, paying special attention to the fact that most of them never expected to have to code as part of their degree. As a psychologist, I recognise the value of identity and community in learning, and I conduct pedagogical research looking at how we can best support students on their university journey. Together with the other editors, we came up with the (somewhat crazy) idea for this book, and I'm delighted to see it turn into reality.

LinkedIn: https://www.linkedin.com/in/kasia-banas/

Work website: https://edwebprofiles.ed.ac.uk/profile/kasia-banas

Umberto Noè (he/him)

Lecturer, School of Philosophy, Psychology and Language Sciences, University of Edinburgh



I am a Lecturer in the Department of Psychology at the University of Edinburgh, where I focus on creating inclusive learning environments in statistics and programming. I care about making technical education accessible to all students, regardless of background. With over a decade of experience, I champion innovative methods like peer programming, i.e. the extension of pair programming to larger groups, to foster engagement, belonging, and collaboration. As editor of and author on this book, I want to share my expertise to support educators and future students.

Work website: https://edwebprofiles.ed.ac.uk/profile/umberto-noe

ORCID: https://orcid.org/0000-0002-4962-5189

Christopher Aldous Oldnall (he/him)

Lecturer, Edinburgh Medical School, University of Edinburgh



I come from a statistics and mathematics background and now teach programming through applied statistics and machine learning in health and social care. Much of my teaching focuses on helping learners get past the fear of starting to code, using small, practical tasks to build confidence early on. I mostly work with students from non-technical backgrounds and place a strong emphasis on learning by doing, often through collaborative work that makes coding feel less isolating and more manageable – an approach that underpins my contribution to this book.

ORCID: https://orcid.org/0000-0003-0160-8109

Personal Website: https://chrisoldnall.github.io/

Serveh Sharifi Far (she/her)

Lecturer in Mathematical Data Science, University of Edinburgh



I am a Lecturer in Mathematical Data Science and a Fellow of the Higher Education Academy. My background is in Statistics, and my research focuses on statistical methodology and its applications in the social and health sciences. I teach undergraduate and postgraduate courses in statistics and data science, including programming in R and Python and working with learners from diverse backgrounds. I am also interested in data science pedagogy and how it can support more inclusive and responsible uses of data, which motivates my contribution to this book as an editor and author.

ORCID: https://orcid.org/0000-0001-8403-6286

Clare Llewellyn-MacRae (she/her)

Senior Lecturer, School of Social and Political Science, University of Edinburgh



I'm a senior lecturer in governance, data, and technology, based in a politics and international relations department. My background is in AI and natural language processing. I use these skills to answer real-world, politically motivated questions that are grounded in theory. My research focuses on how AI and Machine Learning methods can be developed and used to understand political views, identity, and behaviour in massive data sets such as news, social media, and archives. I really enjoy sharing my knowledge of coding, AI, and machine learning, as well as my experimental and procedural skills, to support safer, more accountable technology in public-sector and democratic contexts. I have supported Scottish, UK, EU, and US government stakeholders through data-informed research on AI, digital influence, and policy engagement. I have taught programming to people from very diverse backgrounds at different stages of their careers. I love the teaching experience, I learn so much from it, and I enjoy being in that moment when someone gets something that previously felt impenetrable. There is nothing more rewarding.

Work website: https://www.sps.ed.ac.uk/staff/clare-llewellyn

LinkedIn: https://www.linkedin.com/in/clare-llewellyn-4749318/

Beatrice Alex (she/her)

Professor, School of Mathematical & Computer Sciences, Heriot-Watt University



I'm a professor and researcher in language technology who's spent years extracting meaning out of messy real-world text: clinical notes from hospitals, 19th-century newspapers, historical archives, you name it. My work sits at the meeting point of natural language processing and the humanities, social sciences and healthcare, which means I've taught programming to people who have never looked at a terminal before but are keen to learn. That experience shapes my contribution here: I care about making computational methods feel approachable, especially for learners who don't see themselves as "tech people" but have brilliant questions worth asking with code.

Work website: https://researchportal.hw.ac.uk/en/persons/beatrice-alex/

Work website: https://www.ltg.ed.ac.uk

Ozan Evkaya (he/him)

Lecturer in Statistics, School of Mathematics, University of Edinburgh



I am a Lecturer in Statistics (FHEA) at the School of Mathematics (SoM), University of Edinburgh, teaching statistics and data science at undergraduate and postgraduate levels. My research focuses on statistical modelling and copula applications. Recently, I have been interested in generative AI and its impacts on teaching. I have been involved as SIG lead in "AI in statistics education" through the Researchers of Statistics Education Network (RoSE) and the Academy Network of AI Leads in the Mathematical Sciences. As editor of and author of this book, I am motivated to share my diverse set of experiences with interested readers.

Personal website: https://oevkaya.netlify.app/home

LinkedIn: https://www.linkedin.com/in/ozanevkaya/

Franziska (Ziska) McManus (she/her)

Teaching Assistant, Edinburgh Medical School, University of Edinburgh



I teach introductory R and Python within a data science context to undergraduates, postgraduates, and modern apprentices. Teaching across psychology, medicine, and health and social care contexts allows me to identify differences across disciplines and adapt my teaching for each learner. My background is in psychology - particularly psychology of (human) languages - and people often think that teaching data science is an unusual 'career change'. In other words, the same expectation that surprises many of my students when they sign up for a programme of study and suddenly discover that there is a whole lot of programming to learn along the way. I love the challenges and joys of working with beginner programmers across contexts - helping them to see the relevance of programming, and its use as a tool and as a way of thinking.

I am an editor and the project coordinator for this book.

LinkedIn: https://www.linkedin.com/in/franziska-mcmanus

ORCID: https://orcid.org/0009-0007-7218-8523

Chapter Authors

Samantha J. Ahern (she/her)

Education Co-Lead, Centre for Advanced Research Computing, University College London



Academically I'm a Computer Scientist, but am an experienced education professional specialising in computing and digital research skills. I focus on the social impacts of technology, including social justice and environmental sustainability. My work in education development is focused on the notion of intentional hospitality, and I enjoy helping colleagues develop as educators. I am a Fellow of AdvanceHE and a Senior Certified Member of ALT.

Portfolio: http://reflect.ucl.ac.uk/sahern

Work website: https://profiles.ucl.ac.uk/38404-samantha-ahern

Samantha Alvarez Madrazo (she/her)

Senior Teaching Fellow and Statistical Teaching Coordinator, Imperial College London



I am a pharmacist, health data scientist and educator. Learning programming at the same time as learning statistics can be a challenging journey. Being surrounded by supportive and inspiring colleagues and having a continuous dialogue with online and on-campus postgraduate students enriches my journey. My motivation to contribute to this book is to keep learning from each other, share ideas about empowering students to develop a coding mindset and reflecting about online teaching.

LinkedIn: https://www.linkedin.com/in/samantha-alvarez-b3137211

Neil Anderson

Reader (Associate Professor), Queens University Belfast



I am an educator and researcher in computing with a particular interest in software engineering education, student engagement, and creating effective learning communities. Throughout my career, I have worked to help students develop confidence as programmers and problem-solvers, whether they are learning on campus or through online and distance-learning programmes. My interests span assessment, feedback, learning analytics, and the opportunities and challenges presented by generative AI in education. Alongside this, I have collaborated across disciplines on the development of software systems and digital technologies to support public health and healthcare research. I am passionate about making computing accessible, inclusive, and relevant, helping learners from a wide range of backgrounds discover how programming can be used to solve meaningful real-world problems.

William H. Bell

Teaching Fellow, University of Strathclyde



I have developed software within large international collaborations and commercial projects, using a range of different programming languages and hardware platforms. I starting teaching software development concepts more than two decades ago, including postgraduate and undergraduate students, as well as the general public. I was responsible for many programming articles in the Raspberry Pi magazine and have run public events in France, Switzerland and the UK. I enjoy learning new technologies and delivering useful learning.

Personal website: https://whbell.net/

Work website: https://www.strath.ac.uk/staff/bellwilliamdr/

Yanina Bellini Saibene (she/her)

Community Manager, rOpenSci, RLadies+, Universidad Austral



I work at the intersection of open science, community building, and research software, with a focus on making knowledge more accessible across languages and regions. My background is in information systems and data science, and much of my work centers on supporting communities in the Global South through mentoring, training, and localization efforts. I contributed to this book to share practical lessons from translating teaching materials, highlighting both the challenges and opportunities of creating more inclusive and globally relevant educational resources.

Personal website: https://yabellini.netlify.app/

Github: https://github.com/yabellini

Devanjan Bhattacharya (he/him)

Research Fellow, University of Edinburgh



Devanjan is a Research Fellow on WARSHARE where he leads a research strand aimed at computational data analytics using AI/ML. His areas of research and teaching being geomatics and geo-spatial technologies, he works towards intelligent geospatial analytics merging remotely sensed data, navigation technologies, informatics and GIS together, which are proving a strong platform to progress towards sustainable smart societies, and he continues his endeavour towards that goal, through academic engagements.

In the University of Edinburgh, he has reprised the roles of researcher and teacher. He was a Marie Curie Postdoc Fellow on data-driven innovation under TRAIN@ED project, where he worked with peace-building processes compiling spatial data collected online, using AI, NLP and geo-visualization. Then he was on a Teaching Fellowship at the Bayes Centre, involved in the running of earth observation courses. Presently, he holds a Research Fellowship in an ERC/UKRI Advanced Grant in the School of Social and Political Science, developing language parsing in the domain of conflicts and wars media data.

LinkedIn: https://www.linkedin.com/in/devanjan-bhattacharya-13428b7/

Work website: https://edwebprofiles.ed.ac.uk/profile/devanjan-bhattacharya

Neil Bray (he/him)

Senior Epidemiology Scientist, UK Health Security Agency



My background is in public health, where coding has become an essential skill for my career. I started teaching programming as I wanted others to see the benefits of coding, to feel like it was achievable, and to enjoy the learning process. As I became a more experienced coder, I realised that programming was not just about writing code but also about developing a particular way of thinking. My contribution to this book is a chapter that explores this further and includes strategies that educators can use to develop this coding mindset in their students.

ORCID: https://orcid.org/0009-0006-4315-9486

Alex Buckley

Associate Professor, Learning and Teaching Academy, Heriot-Watt University

Arif Budiarto (he/him)

Postdoctoral Research Fellow, University of Edinburgh



I'm a health data scientist specialising in clinical prediction modelling, EHR analytics, and translational (responsible) AI. After completing my PhD in Medical Informatics at the University of Edinburgh, I moved into a postdoctoral fellowship developing deployable COPD risk_stratification tools for NHS clinicians. I love turning complex health data into tools that can help people.

LinkedIn:
https://www.linkedin.com/in/abudiarto/

Work website:
https://edwebprofiles.ed.ac.uk/profile/arifbudiarto

Yashique Chalil (he/him)

Tutor, Reid School of Music, University of Edinburgh



I am a Software Engineer and Sound Designer dedicated to exploring the limits and possibilities of creative expression. Working at the intersection of music technology, academia, and community-based practices, I aspire to build tools that make this expression more accessible and intuitive. My fascination with the intricate relationship between programming, electronics, nature, culture and the arts inspires me to pursue this path.

Personal website:
https://yashiquechalil.com/

LinkedIn:
https://www.linkedin.com/in/yashique/

Rebecca Chislett (she/her)

Associate Professor, University College London



I am an associate professor in the High Energy Physics group at UCL. Following on from my PhD working on the LHC, my current research focusses on precision muon physics, specifically muon g-2 and Mu2e. Computer programming is integral across particle physics to analyse the large volumes of data and use innovative techniques to produce the best results. This motivates my teaching of both particle physics and (most relevant here) an introductory python programming course for first year undergraduates. The ethos is about coding for practical uses in physics, to aid and complement their learning and hopefully set them up for their future careers.

Work website:
https://profiles.ucl.ac.uk/29547-rebecca-chislett

Lee Clift (he/him)

Teaching Associate, University of Strathclyde



I am a Teaching Associate in Computer and Information Sciences with a background in robotics and a passion for making technology accessible and engaging for everyone. Alongside my university teaching, I run outreach sessions in schools across the United Kingdom, introducing young learners to programming through creative, hands-on activities.

My work focuses on supporting educators in integrating emerging technologies, such as generative AI and robotics, into their teaching practice, as well as on exploring how endangered and minority languages can be sustained in a digital world.

Work website: https://www.strath.ac.uk/staff/cliftleedr/

ORCID: https://orcid.org/0000-0002-8313-9934

Matthew Collins (he/him)

Reader (Associate Professor), Queens University Belfast



I'm a former student who never left the university so they eventually started paying me and made me part of the lecturing staff. I teach a variety of modules across our different undergrad and postgrad pathways on both discrete maths and programming subjects. I have a keen interest in Virtual Reality and run a VR lab where students can produce innovative and immersive final year projects and am always interested in exploring what's possible. Inspiring student's to use their imagination to write creative software that unlocks the potential of new hardware is extremely rewarding.

Rebecca Colquhoun (they/them)

Teaching Fellow, Imperial College London



I am a Teaching Fellow in Geophysics and Quantitative Geoscience, with a particular interest in how we best teach maths and programming to students who may not have directly chosen to study them! I'm also interested in how we make education, including programming education, more accessible and inclusive for everyone.

LinkedIn: https://www.linkedin.com/in/rebeccacolquhoun/

Personal website: https://rebeccacolquhoun.github.io

Chris Cooling

Senior Teaching Fellow,
Imperial College London



As part of the Early Career Researcher Institute, I teach programming courses to early career researchers from all departments across the university. I mainly teach Python, LaTeX, Fortran and AI-Assisted Programming. I have an interest in helping researchers learn the tools and practices that they can immediately use practically to achieve their research goals. I am also interested in the knotty problem of how we can incorporate generative AI into research computing such that we improve efficiency and capability, whilst retaining reliability, accuracy and accountability.

Work website:
https://profiles.imperial.ac.uk/c.cooling10

LinkedIn:
https://www.linkedin.com/in/chris-cooling-09938125/

David Cutting (he/him)

Reader (Associate Professor),
Queen's University Belfast



I'm a software fanatic who fell into education and hasn't (yet) fallen out of it or with it. Programming since single digits of age I've made every technical mistake imaginable and still seem to generate technical debt at an astounding rate. I'm the author of some open source tools, mostly very obscure networking libraries and software. My research interests are around big dirty systems, often of the legacy and distributed variety and somewhat late to the party AI in software development. In my spare time I am an amateur rocketeer slowly working my way through the UK Rocketry Association certification levels with mixed success.

Personal website:
https://davecutting.uk/

LinkedIn:
https://www.linkedin.com/in/davecutting/

Laila Dabab Nahas

Edinburgh Medical School,
University of Edinburgh



I am passionate about programming, teaching, and research. My academic background is in biotechnology, where I completed both my Bachelor's and Master's degrees, followed by a PhD and postdoctoral research focused on programming and research.

I currently teach postgraduate and undergraduate students in Data Science for Health and Social Care. I support students' many of whom come from non-technical backgrounds' in developing programming skills and learning how to approach complex projects. I encourage them to break problems into manageable steps, helping them build confidence and progress effectively.

LinkedIn:
https://www.linkedin.com/in/lailadababnahas/

Frederik Dahl Madsen (he/him)

PhD Student, School of Geosciences, University of Edinburgh & Lyell Centre, British Geological Survey



I am a PhD student at the University of Edinburgh and British Geological Survey, specialising in geomagnetism and geodesy. I am in particular interested in understanding geomagnetic jerks, and how they affect the length-of-day. Outside of my academic endeavours, I am engaged in outreach and engagement around Scotland. I also teach geophysics and science communication at the University of Edinburgh, and am engaged in improving our approach to assessment feedback in our undergraduate degrees.

Personal website: https://fdmadsen.wordpress.com/

ORCID: https://orcid.org/0000-0002-0621-1541

Louise Dash (she/her)

Associate Professor (Teaching), University College London

I'm a teaching academic with a research background in computational condensed matter physics. I've been leading in computing education as part of undergraduate physics degrees for over 15 years. I'm particularly passionate about embedding coding as a core skill within physics degrees, not just in programming classes but as a fundamental part of the wider curriculum. Our students come from a diversity of backgrounds and prior experience with programming, and my approach to teaching is to encourage experimentation and increase confidence for all students.

Work website: https://profiles.ucl.ac.uk/38506-louise-dash/

Mar Doig (they/them)

Engineering Education Lead at Perk (https://www.perk.com)

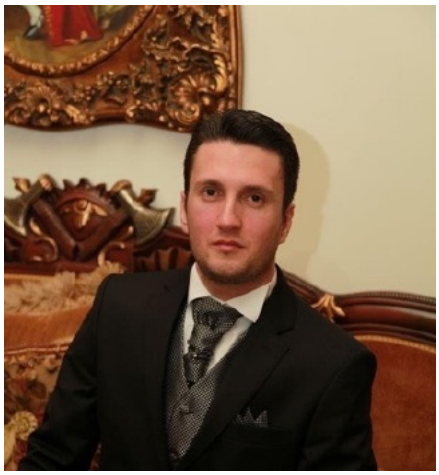


I'm a software engineer and educator focused on the human side of technical literacy. As the Engineering Education Lead at Perk, I designed an in-house academy that turns curiosity into production-ready code, using my own career pivot as a roadmap. Away from the screen, I spend my time exploring high-fantasy worlds and rural Scotland alongside my husband and three children.

LinkedIn: https://www.linkedin.com/in/mardoig/

Joseph El Gemayel

Teaching Fellow, University of Strathclyde



I am an experienced computer science educator and Fellow of the Higher Education Academy, with over a decade of teaching across undergraduate and postgraduate levels, specialising in Artificial Intelligence and Machine Learning. My contribution to this book draws on designing and delivering online modules, creating accessible Python programming instruction for non-computer scientists using Jupyter Notebooks, and developing automated assessment approaches. I am also particularly interested in how AI-driven methods can enhance learning, making complex subjects more engaging, inclusive, and effective for large, diverse cohorts.

Work website:
https://pureportal.strath.ac.uk/en/persons/joseph-el-gemayel/

LinkedIn:
http://www.linkedin.com/in/joseph-elgemayel

Sarah Elliot (she/her)

PhD Candidate , University of Edinburgh

I am a PhD student in Medical Informatics with a strong commitment to lifelong learning. Since joining Edinburgh University I have had the opportunity to teach and support others in their learning journeys, which has been very rewarding. I am passionate about learning, sharing knowledge, and using data that can improve public health and healthcare outcomes.

Tuğrulcan Elmas

Lecturer, University of Edinburgh



I am a Lecturer in the School of Informatics at the University of Edinburgh, directing Social Media Analysis and Support For Humanity (SMASH) Lab and teaching Computational Social Science. My research focuses on social media & misinformation, Human-AI interaction, and automating science & education. Recently, Im particularly interested in developing methods to teach Computational Social Science without overdelegating thinking and planning to AI, which is one reason I am glad to be a contributor to this book.

Personal website:
https://tugrulcan.com/

LinkedIn:
https://uk.linkedin.com/in/tugrulcanelmas

Tom Farrington (he/him)

Senior User Researcher, Storm ID



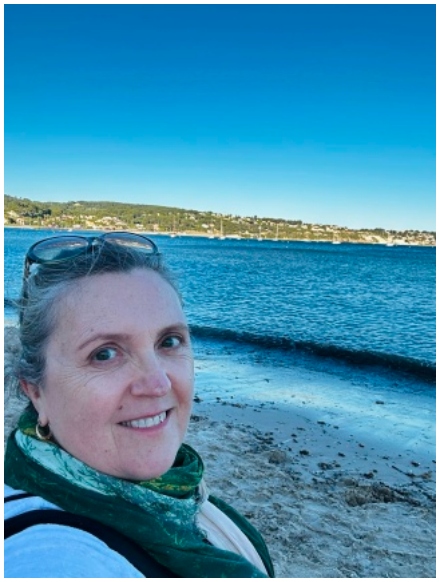
I'm a senior user researcher at Storm ID, a digital consultancy based in Leith, Edinburgh, where I've worked on projects for clients such as the Scottish Government, Historic Environment Scotland, and the Edinburgh Festival Fringe Society. Before this, I lectured and researched at Heriot-Watt University in Edinburgh, where I was Head of Research Methods teaching and worked on a wide variety of research projects in the School of Social Sciences. I've also taught at South East European University in Tetovo and the University of Edinburgh, where I received my undergraduate, Master's, and doctoral degrees, and where I was Co-Director of the Scottish Universities' International Summer School.

In my spare time, I like metal and pop and comedy, writing and reading and talking, and running and thinking.

ORCID: https://orcid.org/0000-0003-4650-2375

Suzanne Goopy

University of Edinburgh



I am a social and visual anthropologist by training and an educationalist by design. In practice, across a career spanning (too) many years and a number of countries, I have sought to find innovative ways to engage with contemporary issues and challenges both in research and education. I have worked with academic and industry research partners and collaborated across different fields including data science, library sciences, environmental design, medicine, nursing, psychology, health, migration, arts, ethics, and documentary practice. I enjoy exploring and teaching others about innovative visual and cultural research and dissemination methodologies. I most especially enjoy the challenges that arise from multilayered explorations of knowledge translation and knowledge exchange.

ORCID: https://orcid.org/0000-0001-7862-6212

Jonathan Holt (he/him)

PhD Student, Imperial College London

I am a PhD student at Imperial College London focusing on the biomechanics of the knee. Most of my work has been on the coding side of the development of simultaneous partial knee reconstructions and ligament repairs.

Github: https://github.com/jjholt

Nick Jayanth

Teaching Fellow, Imperial College London



I am a Teaching Fellow and joined the teaching family as a Graduate Teaching Assistant after my Masters in 2020, and it was the best decision! I enjoy teaching statistics in the public health context. I also teach programming, mainly in R. I want to share my experiences in these fields and would like to learn from others. Beyond my role, I enjoy history and travelling. I often say to myself if I was not a teacher, I would have been on some archaeological dig in Egypt!

ORCID: https://orcid.org/0000-0002-6906-4219

William P. Kay (he/him)

Lecturer in Statistics, Cardiff University | Prifysgol Caerdydd



I trained as an ecologist and, for most of my time as a student, never imagined I would become a Lecturer in Statistics - I was intimidated by both statistics and programming and avoided them whenever possible! Everything changed during my PhD, when my supervisors encouraged me to learn R. What began as a necessity quickly became a passion. I realised that programming was not just a reproducible way to run analyses, but a powerful tool for understanding statistics and tackling the messy, complex data that ecologists work with every day.

Today, I lead statistics teaching across all levels in the School of Biosciences at Cardiff University, and lead the School's digital education strategy. My scholarship focuses on statistics pedagogy; helping students build confidence in statistics and programming through mitigating anxiety, and developing statistics curricula. I am inspired by Steve Jobs' observation that "coding is a mirror to your thought process - it teaches you how to think" and remind my students that learning statistics and programming requires only three things: patience, persistence, and faith.

ORCID: https://orcid.org/0000-0001-6855-7153

Work website: https://profiles.cardiff.ac.uk/staff/kayw

Stuart King

Reader, University of Edinburgh

I am a Reader in applied mathematics, and my research interests span across applied machine learning and mathematical modelling, with applications from health image data through to environmental problems. I have taught programming and data science in various places over the last twenty years, often with an interdisciplinary angle, and often at an introductory level. I'm really interested in introducing more learners to thinking algorithmically and the value that programming can bring as a way of solving problems.

LinkedIn: https://www.linkedin.com/in/stuart-king-a0b64b311

Work website: https://webhomes.maths.ed.ac.uk/~skingg3/

Olexandr Konovalov (he/him)

Lecturer, University of St Andrews



I am a Lecturer in Computer Science and a Fellow of the Software Sustainability Institute. My background is in Pure Mathematics, and initially I came to Scotland from Ukraine to work on the open-source mathematical software system called GAP. Now I teach Python for diverse cohorts of learners and advocate for good practices of working with code and data. I am an Instructor and an Instructor Trainer for The Carpentries, where I have established and coordinate the Ukrainian Carpentries Community. My contribution to this book is very much based on the experience of translating The Carpentries teaching materials into Ukrainian.

Personal website: https://olexandr-konovalov.github.io/

ORCID: https://orcid.org/0000-0001-5299-3292

Patricia A. Loto (she/her)

Teaching Assistant, Software Quality Research Group - UNNE



My background is in Information Systems and I work as a university programming instructor. I have taught people approaching code for the first time, with no prior computing background, and I currently teach core computer science courses — a journey that showed me that, regardless of the level, what defines a good programming class is designing for those who need it most, leaving no one behind.

Personal website: https://patricia-loto.netlify.app/

Github: https://github.com/PatriLoto

Andrew MacLaren (he/him)

Associate Professor, Heriot-Watt University



I have a background in the social sciences, having researched leadership and language in my PhD, and my current research interests focus on teamwork and communication. I see teaching as central in an academic's life and feedback is fundamental to how our students learn. My work on feedback led to my contribution to this book through the Action Feedback Protocol project, which takes a systems approach to delivering feedback. I also co-host a podcast called Teamcraft which offers an academic and a technology leader's views on teamwork – available on all platforms and Youtube.

Personal website: https://sites.google.com/view/theactionfeedbackprotocol/home

LinkedIn: https://www.linkedin.com/in/dr-andrew-maclaren-40518216b/

Tiago A. Marques (he/him)

Principal Research Fellow, CREEM; University of St Andrews / CEAUL; DBA; Faculdade de Ciências da Universidade de Lisboa



I was once a biologist that become more and mor atracted by the dark side of the force, so then I did and MSc and a PhD in statistics. These days I work as an ecological statistician, mostly on the topic of animal abundance estimation. I teach, and have a keen interest in how we should be teaching, statistics to ecologists that might be as afraid from it as I once was.

Work website: https://research-portal.st-andrews.ac.uk/en/persons/tiago-a-marques/

Github: https://github.com/TiagoAMarques

Katerina Michalickova (she/her)

Principal Teaching Fellow, Imperial College London

I'm an educator with a background in high performance computing and bioinformatics. I focus on coordinating and promoting a comprehensive programme in computing and data skills for early-career researchers. I'm always interested in exploring new ways to facilitate learning and in pondering how to adapt our teaching to the LLM generation.

Work website: https://profiles.imperial.ac.uk/k.michalickova

LinkedIn: https://www.linkedin.com/in/katerina-michalickova-a94817141/

Lucia Michielin (she/her)

Digital Skills Training Manager, University of Edinburgh



Lucia Michielin works within the Data-Led Methods and Research Technology team (EFI-University of Edinburgh), where she leads a data-led methods training programme. Her work supports researchers in the Arts, Humanities, and Social Sciences who wish to incorporate digital and data-driven methods into their research projects and offers a wide range of learning opportunities, from introductory coding courses to hands-on workshops, challenge-led collaborative work, and in-depth explorations of various digital methods. Lucia has considerable experience in designing and delivering training courses focused on data skills and digital research methods, as well as applying computational methods in humanities research. She holds a PhD in Classics from the University of Edinburgh and has developed extensive expertise in the Digital Humanities.

Work website: https://edwebprofiles.ed.ac.uk/profile/dr-lucia-michielin

Elaine Mowat

Programme Manager, University of Edinburgh



I am a learning and development specialist who has worked on a wide range of educational initiatives in higher education and the third sector in roles such as learning design, tutoring, coaching and mentoring and programme management. I like working on projects that are values-based and purpose-driven, and enjoy collaborating with others to create experiences that support and challenge learners to discover new possibilities - 'come with us on a learning adventure!'

Tom Mudd

Reader, School of Music, University of Edinburgh

I'm a musician who primarily uses coding as a way to explore music-making. I teach on courses that relate to this in the school of music at the University of Edinburgh.

Personal website: https://tommudd.co.uk

Maeve Murphy Quinlan (she/her)

Research Software Engineer, University of Leeds

I'm a research software engineer with a background in planetary science research. I teach researchers how to implement sustainable software practices, in a way that aims to be welcoming, encouraging, and inclusive. When I'm not researching or coding, you can find me playing a TTRPG with my friends: the perfect place to live the dream of redistributing wealth and getting a solid eight hours of uninterrupted sleep. While I helped to brainstorm and plan our chapter in this book, the writing credit must firmly sit at the feet of my fantastic lead co-author, Francisco!

Personal website: https://murphyqm.github.io/

ORCID: https://orcid.org/0000-0003-2958-1008

Lisa Otty

University of Edinburgh



I am Director of the Centre for Data, Culture & Society and Head of Data-Led Methods and Research Technology at the Edinburgh Futures Institute, where my work focuses on finding creative and engaging ways to support researchers exploring, adopting and adapting data-led and applied computational methods. My background is in Digital Humanities and textual scholarship, I am motivated by the belief that arts and humanities scholars can bring invaluable insights to the design, development and implementation of tools and technologies.

Work website:
https://edwebprofiles.ed.ac.uk/profile/dr-lisa-otty

Elizabeth Pankratz (she/her)

Lecturer, University of Edinburgh



I'm a linguist turned statistics teacher. My teaching practice is built on ideas from critical pedagogy and psychological research on motivation, learning, and memory. I care a lot about helping people overcome their fear of statistics and programming, and I think that a key way to do this is to give people the freedom to take risks and try new things without penalty, an idea that I explore in my chapter of this book.

Personal website:
https://elizabethpankratz.github.io

Github:
https://github.com/elizabethpankratz

Olga Petrovska

Institute of Coding in Wales, Swansea University



I am a researcher and educator at the Institute of Coding in Wales and Swansea University, and a Senior Fellow of the Higher Education Academy. My academic background is diverse, spanning linguistics, pedagogy, and theoretical computer science. I am deeply interested in how people learn, reason about, and communicate technical ideas. My recent work focuses on computing education, with particular emphasis on Generative AI literacy.

Personal website:
https://opetrovska.github.io/#about

LinkedIn:
https://www.linkedin.com/in/olgapetrovska/

Ruini Qu (she/her)

University of Edinburgh



I specialize in Management Science and Operations Research, focusing on how technical tools can be leveraged within Business Analytics. As someone relatively new to teaching code, I am constantly exploring different pedagogical approaches to find what truly resonates with business students. My goal is to move beyond syntax and help learners see the practical side of programming. Outside the classroom, I enjoy DIY projects and crafting handmade pieces—finding the same joy in building things by hand as I do in building a good piece of analysis.

Nguyen Quang Chien (he/him)

Research Associate, University of Edinburgh

I joined the University of Edinburgh, School of Engineering, on a contract-based work related to software development. My background is in civil engineering, more precisely, hydraulics, and I had university teaching experience in Vietnam before Microsoft Copilot and Google Gemini were introduced. Now, civil engineering students are largely accessible to code and data repositories as well as good practices in programming. I believe that a shift in mindset towards programming would help them start their careers strong, and this initiates my contribution to the book.

Personal website: https://coastal-study.uk/

ORCID: https://orcid.org/0000-0002-1952-4360

Francisco Queiroz (he/him)

Lecturer, University of Leeds



I am a designer and educator (not necessarily in that order). My teaching background spans over nineteen years, two countries, and disciplines as diverse as image editing, game prototyping, research methods, and communication. My contribution to this book reflects my intention of making learning a truly collaborative and progressive experience.

Work website: https://ahc.leeds.ac.uk/design/staff/2233/dr-francisco-queiroz

ResearchGate: https://www.researchgate.net/profile/Francisco-Queiroz

Karim Rivera-Lares (she/her)

Teaching Coordinator, University of Edinburgh



I am a psychologist who fell in love with programming. Learning to code was really daunting, but the sense of achievement I felt after solving a problem was thrilling. During my PhD in psychology at the University of Edinburgh, I realised I was my happiest self whenever I was coding and teaching how to code. My favourite thing at work is to see the excitement in students' faces when they solve a problem that they were struggling with. I'm very excited about this book, because I am confident it will contribute to more students getting excited about programming.

LinkedIn: https://www.linkedin.com/in/karim-rivera-lares/

Leo Riviera (he/they)

Innovation & Outreach Lead, Voxa



I'm a software engineer by trade, with a focus on helping cool people do cool stuff. I work at a space startup, organise engineering community groups and produce installations for creative technology festivals. I'm also working on a book about how building opinionated tooling can help startups succeed. When I'm not busy touching computers, I mentor people from underrepresented backgrounds so they develop the skills they need to work as software engineers. I also speak at conferences and contribute to open source, advocating for decentralisation, open data and personal data sovereignty. I'm a big believer in the power of using systems thinking to build a world where hardware, software and peopleware work well together. In that vein, I've contributed to and edited this book.

Personal website: https://cowsay.io

ORCID: https://orcid.org/0009-0004-3316-1852

Gule Saman (she/her)

Associate Professor, Heriot-Watt University Edinburgh



I am an Associate Professor in Data Science with a background in Computer Vision at Heriot-Watt University (HWU), where I serve as Co-Programme Director of the Graduate Apprenticeship in Data Science Programme and Associate Academic Dean. I am passionate about creating inclusive and supportive environments in STEM and co-lead the HWU STEM Inclusivity Network, which was Highly Commended in HWU's Equality, Diversity and Inclusion Champion Team Award 2025.

With a background in STEM, and often being the only woman in my engineering lectures, I understand the importance of fostering inclusivity, belonging, and representation. As a Senior Fellow of the Higher Education Academy and recipient of a commendation in the HWU Inspiring Learning Early Career Award (2021), I am committed to empowering the next generation and contributing to meaningful conversations that help all students recognise and celebrate their place within the STEM ecosystem.

LinkedIn: https://www.linkedin.com/in/gule-saman-389b9ba1/

Ignacio Sepulveda (he/him)

Lecturer, Scotland's Rural College (SRUC)



I'm a chemical engineer doing research and teaching in wildlife and conservation management. My mum used to work as a software engineer, so there were computers around in my house for as long as I can remember. I had my first PC running early versions of DOS and Windows, and I used to invite friends over to play games.

Back at University I learned how to use R and different GIS software. I still remember that feeling, almost like early humans discovering fire.

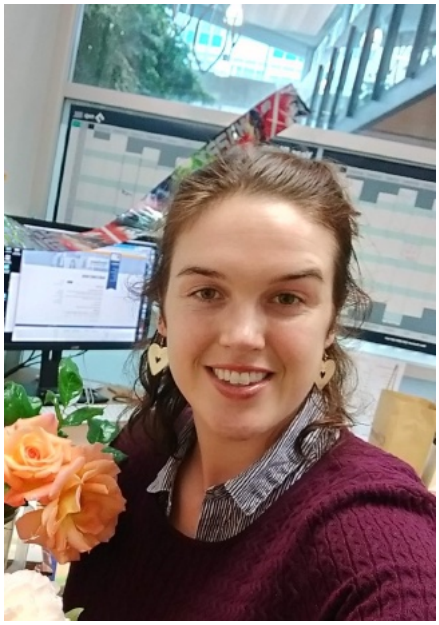
For the past six years, I've been lucky enough to teach conservation students how to discover these skills for themselves. That journey led me to contribute my own experience to this book.

LinkedIn: https://www.linkedin.com/in/ignacio-s-6bb90841/

Work website: https://www.sruc.ac.uk/course-catalogue/wildlife-and-conservation-management/hnc-wildlife-and-conservation-management/

Rebecca Sewell (she/her)

Data Integrity Administrator, Health New Zealand | Te Whatu Ora



Leila Shila Shafti

Teaching Fellow, University of Strathclyde

I have been teaching programming for more than two decades in Spain and the UK. I have experience teaching both online and in-person classes across undergraduate and postgraduate programmes. I am particularly passionate about exploring innovative methods to enhance teaching and learning, with the aim of making complex concepts more accessible and strengthening practical understanding. I focus on creating an inclusive and supportive learning environment that encourages engagement and curiosity.

Hebatallah Shoukry

Assistant Professor, Heriot-Watt University



Hebatallah Shoukry earned her PhD from Heriot-Watt University in 2020 and currently holds the position of Assistant Professor in the Electrical, Electronics, Computer Engineering (EECE) department within the School of Engineering and Physical Science (EPS). Specialising in mathematical and introductory computer science courses for the Data Science Graduate Apprenticeship programme, she emphasises student-staff collaboration to boost student engagement. Recognized for her dedication, she received a commendation at the Teaching Excellence Awards 2021 and attained Fellowship of the Higher Education Academy (FHEA) in 2022 for her commitment to teaching and learning standards.

Focused on student engagement and active learning, Heba Shoukry's research revolves around effective pedagogies to enrich student experiences. Guided by a teaching philosophy centered on fostering strong relationships and communication, she strives to support students both academically and socially.

LinkedIn: https://www.linkedin.com/in/hebatallah-shoukry-799134196

Jeremy Singer

Professor, University of Glasgow



I teach programming to students from various backgrounds - some are Computer Science majors, many are not. My favourite programming languages are Haskell, Java and Rust. I also confess to knowing C and Python. I love seeing students' eyes light up when they 'get it' and understand what's wrong with their code and how to fix it.

Personal website: https://www.dcs.gla.ac.uk/~jsinger

Github: https://github.com/jeremysinger

Sam Skipsey (he/they)

Research Fellow, University of Glasgow

Officially I'm a physicist, and stumbled into research-only University track, but increasingly the things I care about most are; ethics in research and academia (and beyond), teaching people to think better and wider, and computing as a discipline that broadens capability for all. I am also one of the organising committee for the Roller Derby World Cup (and at some point will actually finish the paper on this that we've been co-authoring for more than a year...)

Work website: https://www.gla.ac.uk/schools/physics/staff/samuelskipsey/

Luke Stephens (he/him)

Research Fellow, University of Edinburgh



I'm a Research Fellow in the Neuropolitics Lab at the University of Edinburgh, where I study narrative, foreign policy, and International Relations theory. My work brings together interpretive narrative analysis and computational methods, looking at how the stories we tell shape political identity. I recently worked on a framework for using Large Language Models to perform qualitative narrative analysis, and I'm interested in teaching others to use these tools thoughtfully, without losing interpretive rigour.

Work website: https://www.sps.ed.ac.uk/staff/luke-stephens

Gordon Stewart (he/him)

ScotGrid Technical Co-ordinator, School of Physics and Astronomy, University of Glasgow

I am a research IT specialist who has worked in various disciplines across science and engineering; I currently manage large computer systems primarily used to crunch numbers for CERN, the European Organisation for Nuclear Research. Having originally trained as a software engineer, I now teach physicists and students from other backgrounds how to use computers—particularly those running Linux operating systems—and how to develop software for use in scientific research.

Chris Sutherland (he/him)

Reader, University of St Andrews



I work at the intersection of spatial ecology and statistics, developing biologically realistic models to understand how space use, dispersal, and demography respond to density, environment, and landscape structure. The goal is to turn ecological theory into practical tools for conservation and management.

Personal website: https://sutherlandecology.com/index.html

Mike Taverne (he/him or they/them)

Research Fellow in Electrical Engineering, Northumbria University



With a background in electrical engineering, I am specifically interested in photonics, quantum technologies, and developing related software. Making these complex concepts accessible and opening doors for others to discover the wonders of these subjects has long been a passion of mine. I enjoy teaching and inspiring others. I consider alternative learning and teaching techniques through playful means, such as music and games. I kept this creative approach in mind when writing my contribution to this book.

Work website: https://www.northumbria.ac.uk/about-us/our-staff/t/mike-taverne

ORCID: https://orcid.org/0000-0003-0521-8327

Antonia Voigt (she/her)

Lecturer, Surrey Business School, University of Surrey



My work is driven by a vision of a flourishing future for and through higher education. In the classroom, I am passionate about creating authentic learning experiences that empower students to develop lifelong and lifewide skills. Shaped by my fascination with complex systems, my teaching centres on helping students to find joy in the unpredictable nature of learning. Beyond the classroom, I draw on my background in education and management to support universities as organisations in cultivating educational quality in more sustainable ways.

ORCID: https://orcid.org/0000-0002-0608-2575

Lizzie Wadsworth (she/her)

Lecturer in Biological Sciences, Kingston University



I feel very fortunate to have been taught Python programming during my undergraduate degree, as it is so useful in Biological Sciences and has led to my PhD and career opportunities. I'm passionate about passing this on to the next generation of biologists as I believe this opens so many doors for them. I also find programming to be the most enjoyable subject to teach as it's so hands-on, and watching students go from very little prior background to confidently writing and debugging their own code is very satisfying.

Work website: https://www.kingston.ac.uk/about/staff/dr-lizzie-wadsworth

LinkedIn: https://www.linkedin.com/in/lizzie-wadsworth-670b04170/

Steven Watterson

Lecturer in Computational Biology, the University of Ulster

Ben Waugh (he/him)

UCL



My training was in physics, and I am still a physicist, but my focus has shifted to computing and teaching. I see assessment as a vital part of education, but I don't like marking, so I am constantly striving for approaches that guide learners towards better understanding, and fairly reflect their achievements, without sucking the joy out of life for markers or students.

Work website: https://profiles.ucl.ac.uk/36006-ben-waugh

LinkedIn: https://www.linkedin.com/in/ben-waugh-148053152/

John Wilson

Teaching Fellow, Edinburgh Medical School, University of Edinburgh

I worked in outdoor education for over 10 years where I helped terrified people understand that walking backwards off a cliff was scary, but safe. I later went to university to study ecology, and eventually found myself explaining that although R was scary, it was safe. I have spent the years since helping people learn technical subjects am convinced that people describe things as "scary" or "difficult" they really just mean "unfamiliar".

Eileen Y. Xu (she/they)

University of Edinburgh



I am a mental health researcher and statistics tutor on undergraduate and postgraduate data analysis courses in Psychology. As a tutor, I help students to build confidence in their coding abilities during weekly labs by providing not only technical support, but also encouragement and reassurance. After all, nobody really memorises code - it just gets easier with time, practise and a lot of trial and error!

ORCID: https://orcid.org/0009-0003-5044-1437

LinkedIn: https://www.linkedin.com/in/eileenyxu/

Robert S Young

Senior Lecturer, Edinburgh Medical School, University of Edinburgh; Zhejiang University - University of Edinburgh (ZJE) Institute



I teach various courses in programming, statistics and genomics. I lead a course which introduces Python programming to students from both a biomedical sciences and a biomedical informatics undergraduate degree programme. I enjoy showing students that the computer can be a powerful tool for research, and not something to be afraid of.

I also lead a research programme which investigated how variation in medical traits and disease risk is driven by variation in the genomic content of noncoding, regulatory loci such as promoters and enhancers.

Work website: https://edwebprofiles.ed.ac.uk/profile/rob-young

LinkedIn: https://www.linkedin.com/in/rob-young-9804a780/

Image Register

Chapter: Where Do I Even Start With Pair Programming in My Classroom? A Conversation With Seasoned Practitioners

Figure 2.1

Licence: CC BY-NC-SA

Creator: Pawel Orzechowski

Source: Author's own



Figure 2.2

Licence: CC BY-NC-SA

Creator: Pawel Orzechowski

Source: Author's own



Figure 2.3

Licence: CC BY-NC-SA

Creator: Pawel Orzechowski

Source: Author's own

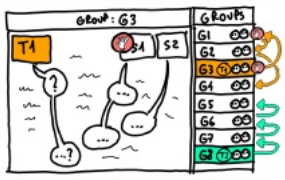


Figure 2.4

Licence: CC BY-NC-SA

Creator: Pawel Orzechowski

Source: Author's own



Chapter: Structured Group Work With Assigned Asymmetrical Roles and Switching: Lessons From Pair Programming Across Disciplines

Figure 3.1

Licence: CC BY-NC-SA

Creator: Pawel Orzechowski

Source: Author's own

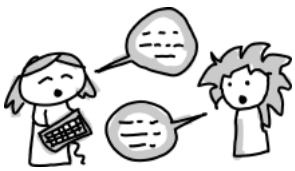


Figure 3.2

Licence: CC BY-NC-SA

Creator: Pawel Orzechowski

Source: Author's own



Figure 3.3

Licence: CC BY-NC-SA

Creator: Pawel Orzechowski

Source: Author's own



Figure 3.4

Licence: CC BY-NC-SA

Creator: Pawel Orzechowski

Source: Author's own



Chapter: It Depends: How to Develop Judgement in Programming

Figure 7.1

Licence: CC BY-NC-SA

Creator: John Wilson

Source: Author's own

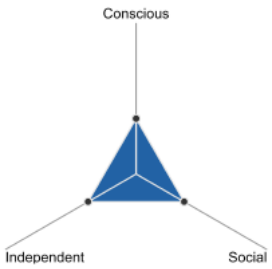


Figure 7.2

Licence: CC BY-NC-SA

Creator: John Wilson

Source: Author's own

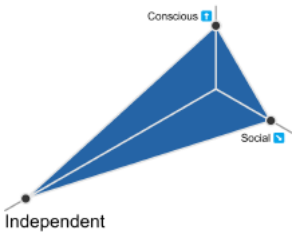


Figure 7.3

Licence: CC BY-NC-SA

Creator: John Wilson

Source: Author's own



Figure 7.4

Licence: CC BY-NC-SA

Creator: John Wilson

Source: Author's own

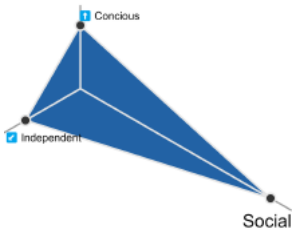


Figure 7.5

Licence: CC BY-NC-SA

Creator: John Wilson

Source: Author's own

Notes: online only

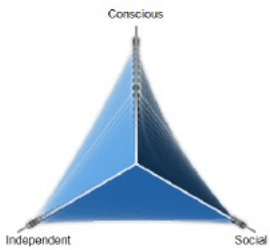


Figure 7.5

Licence: CC BY-NC-SA

Creator: John Wilson

Source: Author's own

Notes: print only

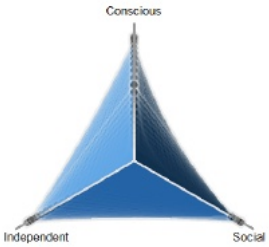


Figure 7.6

Licence: CC BY-NC-SA

Creator: Seren Wilson

Source: Author's own



Chapter: Hype and The Need for Responsible Compute

Figure 8.1

Licence: CC BY-NC-SA

Creator: United Nations

Source: https://sdgs.un.org/goals/goal3

Notes: For details of copyright, see: https://www.un.org/sustainabledevelopment/wp-content/uploads/2019/01/SDG_Guidelines_AUG_2019_Final.pdf



Figure 8.2

Licence: CC BY-NC-SA

Creator: United Nations

Source: https://sdgs.un.org/goals/goal4

Notes: For details of copyright, see: https://www.un.org/sustainabledevelopment/wp-content/uploads/2019/01/SDG_Guidelines_AUG_2019_Final.pdf



Figure 8.3

Licence: CC BY-NC-SA

Creator: United Nations

Source: https://sdgs.un.org/goals/goal8

Notes: For details of copyright, see: https://www.un.org/sustainabledevelopment/wp-content/uploads/2019/01/SDG_Guidelines_AUG_2019_Final.pdf



Figure 8.4

Licence: CC BY-NC-SA

Creator: United Nations

Source: https://sdgs.un.org/goals/goal10

Notes: For details of copyright, see: https://www.un.org/sustainabledevelopment/wp-content/uploads/2019/01/SDG_Guidelines_AUG_2019_Final.pdf



Figure 8.5

Licence: CC BY-NC-SA

Creator: United Nations

Source: https://sdgs.un.org/goals/goal12

Notes: For details of copyright, see: https://www.un.org/sustainabledevelopment/wp-content/uploads/2019/01/SDG_Guidelines_AUG_2019_Final.pdf



Figure 8.6

Licence: CC BY-NC-SA

Creator: United Nations

Source: https://sdgs.un.org/goals/goal16

Notes: For details of copyright, see: https://www.un.org/sustainabledevelopment/wp-content/uploads/2019/01/SDG_Guidelines_AUG_2019_Final.pdf



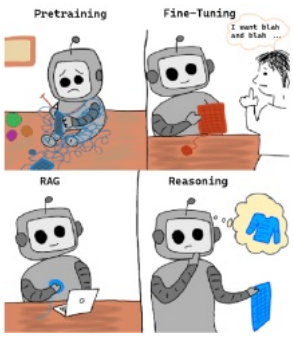
Chapter: An Optimistic Outlook on Teaching, Learning and Assessment for Coding With the Emergence of Generative AI

Figure 9.1

Licence: CC BY-NC-SA

Creator: Laila Dabab Nahas

Source: Author's own



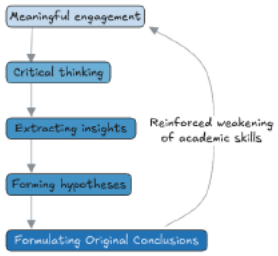
Chapter: AI, Voice, and Style in Programming Education

Figure 10.1

Licence: CC BY-NC-SA

Creator: John Wilson

Source: Author's own



Chapter: Computer Says No: Reducing Technical Barriers to Help Novice Programmers

Figure 12.1

Licence: CC BY-NC-SA

Creator: Lizzie Wadsworth

Source: Author's own

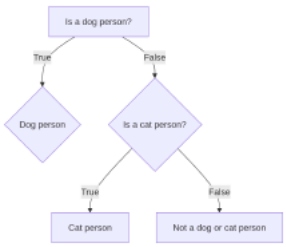


Figure 12.2

Licence: CC BY-NC-SA

Creator: Lizzie Wadsworth

Source: Author's own

Notes: online only

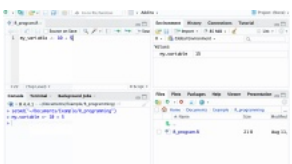


Figure 12.2

Licence: CC BY-NC-SA

Creator: Lizzie Wadsworth

Source: Author's own

Notes: print only

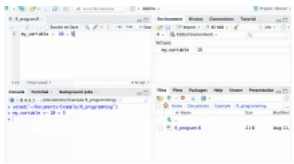


Figure 12.3

Licence: CC BY-NC-SA

Creator: Lizzie Wadsworth

Source: Author's own



Figure 12.4

Licence: CC BY-NC-SA

Creator: Lizzie Wadsworth

Source: Author's own

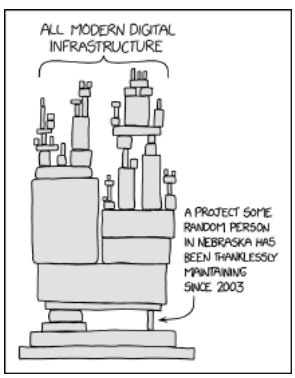


Figure 12.5

Licence: CC BY-NC-SA

Creator: xkcd

Source: https://xkcd.com/



Chapter: Practices To Foster Inclusion and Accessibility in Programming Teaching

Figure 13.1

Licence: CC BY-NC-SA

Creator: Pawel Orzechowski

Source: Author's own

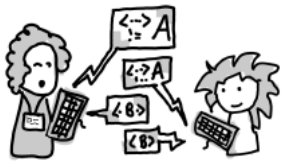


Figure 13.2

Licence: CC BY-NC-SA

Creator: Pawel Orzechowski

Source: Author's own



Figure 13.3

Licence: CC BY-NC-SA

Creator: Pawel Orzechowski

Source: Author's own



Figure 13.4

Licence: CC BY-NC-SA

Creator: Pawel Orzechowski

Source: Author's own

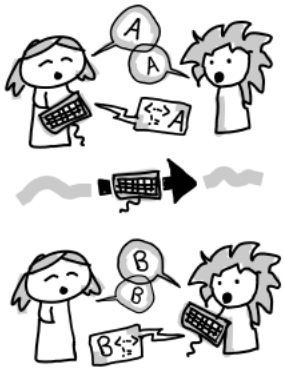


Figure 13.5

Licence: CC BY-NC-SA

Creator: Pawel Orzechowski

Source: Author's own



Figure 13.6

Licence: CC BY-NC-SA

Creator: Pawel Orzechowski

Source: Author's own



Chapter: Overcoming Coding Anxiety: Lowering the Stakes and Making It Fun

Figure 14.1

Licence: CC BY-NC-SA

Creator: The Square Comic

Source: https://www.thesquarecomics.com/

Notes: Via https://www.reddit.com/user/yuva-krishna-memes/

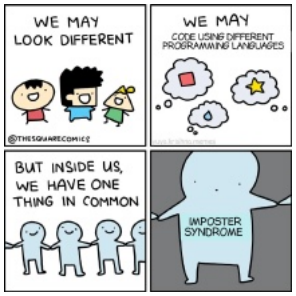


Figure 14.2

Licence: CC BY-NC-SA

Creator: Maria Zacarias

Source: Adapted from https://www.mentalyc.com/

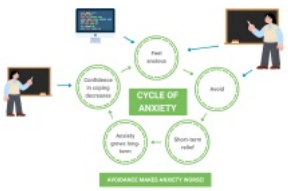


Figure 14.3

Licence: CC BY-NC-SA

Creator: Pratik Gupte

Source: Reused with the author's permission.

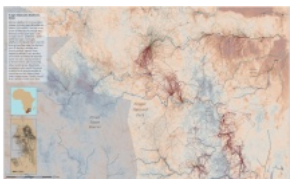


Figure 14.4

Licence: CC BY-NC-SA

Creator: Tiago Marques

Source: Author's own



Figure 14.5

Licence: CC BY-NC-SA

Creator: Maria Zacarias

Source: Author's own



Chapter: 3 Stars and 1 Wish: Small and Frequent Student Reflections Promote a Sense of Wonder and a Community of Vulnerability

Figure 15.1

Licence: CC BY-NC-SA

Creator: Pawel Orzechowski

Source: Author's own

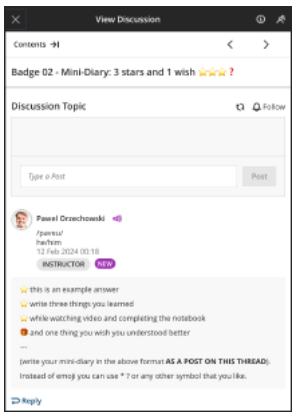
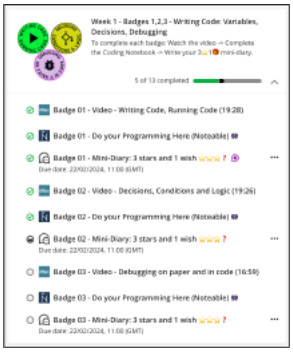


Figure 15.2

Licence: CC BY-NC-SA

Creator: Pawel Orzechowski

Source: Author's own



Chapter: Peer Programming in Action: Pair Programming in Larger Groups

Figure 18.1

Licence: CC BY-NC-SA

Creator: Umberto Noe

Source: Author's own

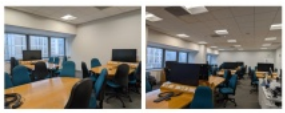


Figure 18.2

Licence: CC BY-NC-SA

Creator: Eileen Xu

Source: Author's own



Chapter: Leveraging the Heterogeneity: Teaching Computing

Skills to a Multidisciplinary Cohort With a Variable Skill Level

Figure 19.1

Licence: CC BY-NC-SA

Creator: Jonathan Holt

Source: Author's own

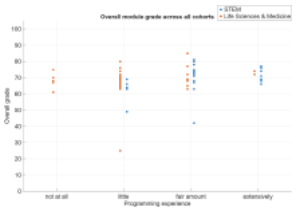
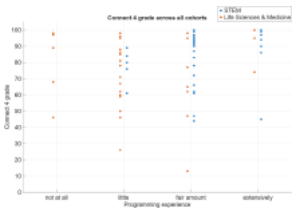


Figure 19.2

Licence: CC BY-NC-SA

Creator: Jonathan Holt

Source: Author's own



Chapter: Learning Together Across Modes: Online and On-Site Pair Programming in a Fusion Course

Figure 20.1

Licence: CC BY-NC-SA

Creator: Pawel Orzechowski

Source: Author's own



Figure 20.2

Licence: CC BY-NC-SA

Creator: Pawel Orzechowski

Source: Author's own



Figure 20.3

Licence: CC BY-NC-SA

Creator: Pawel Orzechowski

Source: Author's own



Figure 20.4

Licence: CC BY-NC-SA

Creator: Pawel Orzechowski

Source: Author's own

Notes: online only



Figure 20.4

Licence: CC BY-NC-SA

Creator: Pawel Orzechowski

Source: Author's own

Notes: print only



Figure 20.5

Licence: CC BY-NC-SA

Creator: Pawel Orzechowski

Source: Author's own

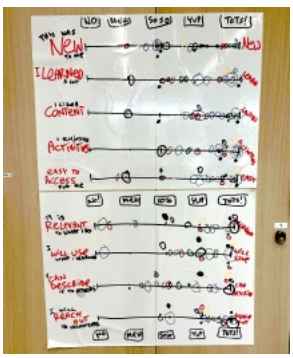
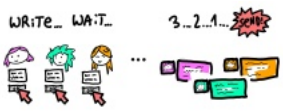


Figure 20.6

Licence: CC BY-NC-SA

Creator: Pawel Orzechowski

Source: Author's own



Chapter: Notebook for Novices? Pros and Cons of Jupyter

Figure 22.1

Licence: CC BY-NC-SA

Creator: Joseph El Gemayel

Source: Author's own



Figure 22.2

Licence: CC BY-NC-SA

Creator: Joseph El Gemayel

Source: Author's own

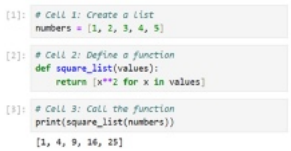


Figure 22.3

Licence: CC BY-NC-SA

Creator: Joseph El Gemayel

Source: Author's own

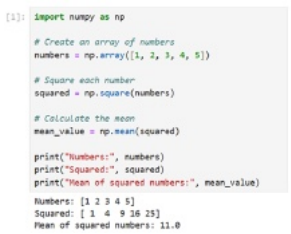


Figure 22.4

Licence: CC BY-NC-SA

Creator: William Bell

Source: Author's own



Figure 22.5

Licence: CC BY-NC-SA

Creator: William Bell

Source: Author's own



Figure 22.6

Licence: CC BY-NC-SA

Creator: William Bell

Source: Author's own



Chapter: Removing Barriers by Programming Without Computers

Figure 25.1

Licence: CC BY-NC-SA

Creator: David Cutting

Source: Author's own



Chapter: Practical Approaches to Using Sound and Music in Programming Pedagogy

Figure 26.1

Licence: CC BY-NC-SA

Creator: Matthew Hamilton

Source: Author's own

Notes: online only

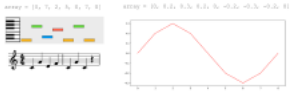


Figure 26.1

Licence: CC BY-NC-SA

Creator: Matthew Hamilton

Source: Author's own

Notes: print only

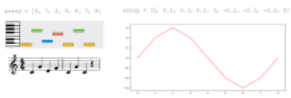


Figure 26.2

Licence: CC BY-NC-SA

Creator: Matthew Hamilton

Source: Author's own

Notes: print only

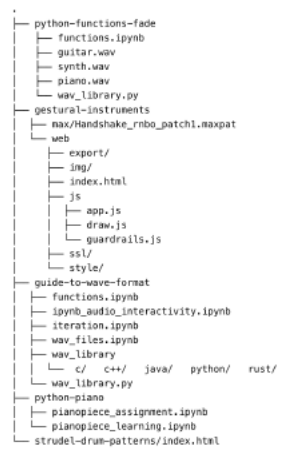


Figure 26.3

Licence: CC BY-NC-SA

Creator: Matthew Hamilton

Source: Author's own



Figure 26.4

Licence: CC BY-NC-SA

Creator: Matthew Hamilton

Source: Author's own

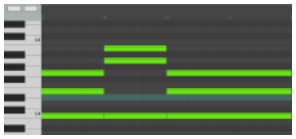


Figure 26.5

Licence: CC BY-NC-SA

Creator: Tom Mudd

Source: Author's own



Figure 26.6

Licence: CC BY-NC-SA

Creator: Charlotte Desvages

Source: Author's own



Figure 26.7

Licence: CC BY-NC-SA

Creator: Tom Mudd

Source: Author's own



Figure 26.8

Licence: CC BY-NC-SA

Creator: Tom Mudd

Source: Author's own

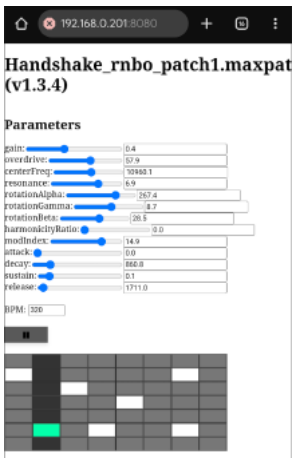
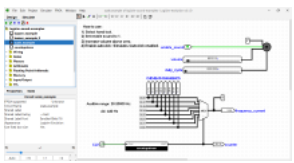


Figure 26.9

Licence: CC BY-NC-SA

Creator: Mike Taverne

Source: Author's own



Chapter: Seeing Before Coding, Doodling Before Doing: How Teaching Data Visualisation Transforms the Way Students Think

Figure 28.1

Licence: CC BY-NC-SA

Creator: Suzanne Goopy

Source: Author's own



Figure 28.2

Licence: CC BY-NC-SA

Creator: Suzanne Goopy

Source: Author's own



Figure 28.3

Licence: CC BY-NC-SA

Creator: Suzanne Goopy

Source: Author's own



Figure 28.4

Licence: CC BY-NC-SA

Creator: Suzanne Goopy

Source: Author's own

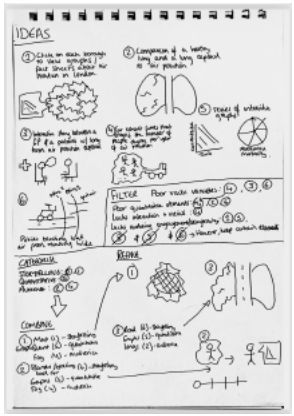


Figure 28.5

Licence: CC BY-NC-SA

Creator: Suzanne Goopy

Source: Author's own



Figure 28.6

Licence: CC BY-NC-SA

Creator: Suzanne Goopy

Source: Author's own

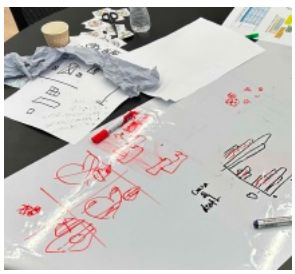


Figure 28.7

Licence: CC BY-NC-SA

Creator: Suzanne Goopy

Source: Author's own

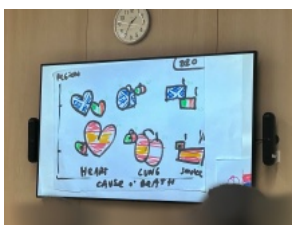


Figure 28.8

Licence: CC BY-NC-SA

Creator: Suzanne Goopy

Source: Author's own

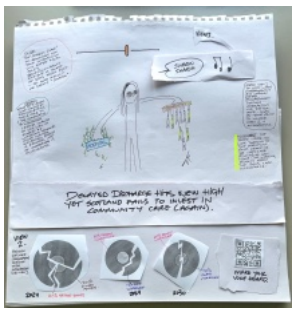
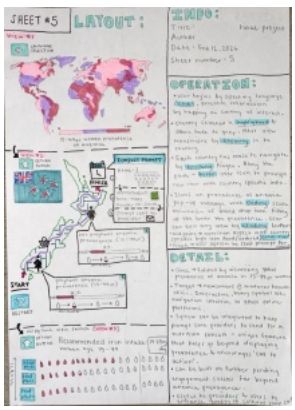


Figure 28.9

Licence: CC BY-NC-SA

Creator: Suzanne Goopy

Source: Author's own



Chapter: Developing a Skills-Based Framework for Assessments in Programming Courses

Figure 30.1

Licence: CC BY-NC-SA

Creator: Elizabeth Pankratz

Source: Author's own



Chapter: Managing the Rubik s Cube of Assessment: The Action Feedback Protocol

Figure 31.1

Licence: CC BY-NC-SA

Creator: Andrew MacLaren

Source: Image adapted and further modified from Rubik s cube v3 by User:Booyabazooka and User:Meph666, modified by User:Niabot, licensed under CC BY-SA 3.0. https://creativecommons.org/licenses/by-sa/3.0/



Chapter: Using Automated Marking in Programming Courses To Enhance Learning

Figure 34.1

Licence: CC BY-NC-SA

Creator: Leila Shafti

Source: Author's own

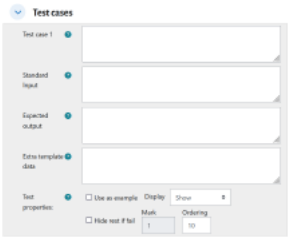


Figure 34.2

Licence: CC BY-NC-SA

Creator: Leila Shafti

Source: Author's own

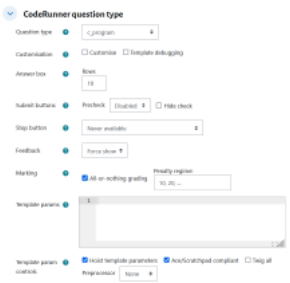


Figure 34.3

Licence: CC BY-NC-SA

Creator: Leila Shafti

Source: Author's own

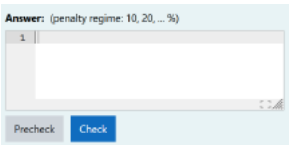
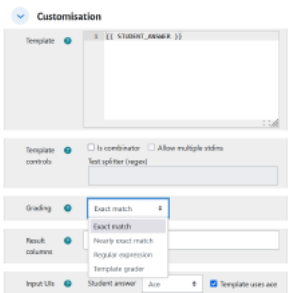


Figure 34.4

Licence: CC BY-NC-SA

Creator: Leila Shafti

Source: Author's own



Chapter: A Gentle Introduction to Coding: Narratives From a Programming Summer School for Social Sciences and Humanities

Figure 36.1

Licence: CC BY-NC-SA

Creator: Pawel Orzechowski

Source: Author's own



Figure 36.2

Licence: CC BY-NC-SA

Creator: Pawel Orzechowski

Source: Author's own



Figure 36.3

Licence: CC BY-NC-SA

Creator: Pawel Orzechowski

Source: Author's own



Figure 36.4

Licence: CC BY-NC-SA

Creator: Pawel Orzechowski

Source: Author's own

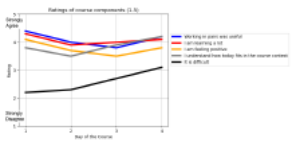


Figure 36.5

Licence: CC BY-NC-SA

Creator: Pawel Orzechowski

Source: Author's own



Figure 36.6

Licence: CC BY-NC-SA

Creator: Pawel Orzechowski

Source: Author's own



Figure 36.7

Licence: CC BY-NC-SA

Creator: Pawel Orzechowski

Source: Author's own

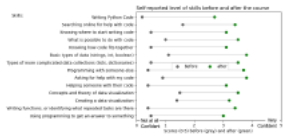


Figure 36.8

Licence: CC BY-NC-SA

Creator: Pawel Orzechowski

Source: Author's own

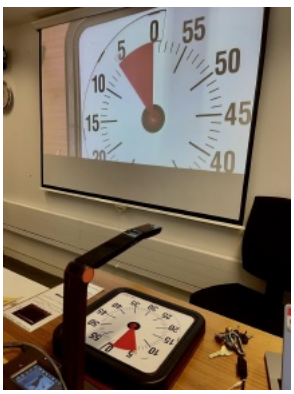
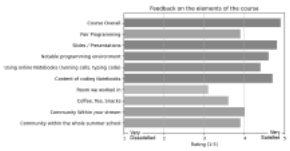


Figure 36.9

Licence: CC BY-NC-SA

Creator: Pawel Orzechowski

Source: Author's own



Chapter: Case Study From Scotland s Rural College: The Wild(Life) Side of Pair Programming

Figure 37.1

Licence: CC BY-NC-SA

Creator: Ignacio Sepulveda

Source: Author's own

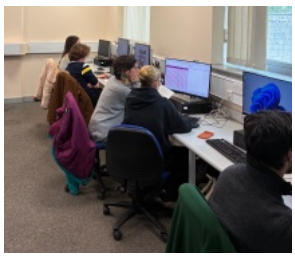


Figure 37.2

Licence: CC BY-NC-SA

Creator: Ignacio Sepulveda

Source: Author's own

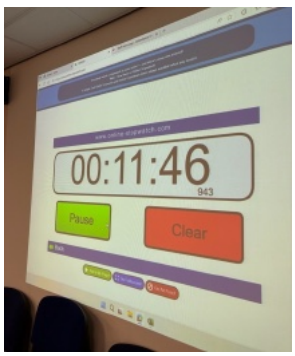


Figure 37.3

Licence: CC BY-NC-SA

Creator: Ignacio Sepulveda

Source: Author's own



Chapter: Changing Civil Engineering Students Mindset Toward Programming

Figure 39.1

Licence: CC BY-NC-SA

Creator: Nguyen Quang Chien

Source: Author's own

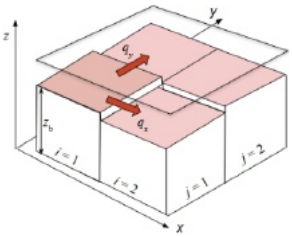


Figure 39.2

Licence: CC BY-NC-SA

Creator: Nguyen Quang Chien

Source: Author's own

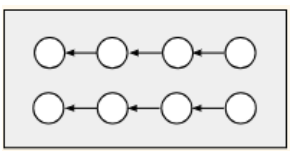
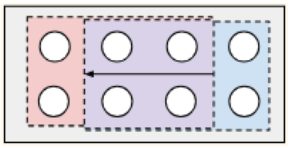


Figure 39.3

Licence: CC BY-NC-SA

Creator: Nguyen Quang Chien

Source: Author's own



Chapter: Lost in Translation: Complexities and Good Practices of Translating Coding-Based Teaching Materials Across Multiple Human Languages

Figure 40.1

Licence: CC BY-NC-SA

Creator: Yanina Bellini Saibene

Source: Author's own

Notes: online only

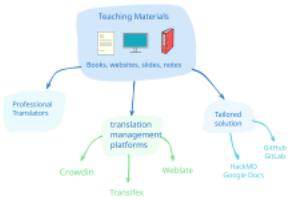


Figure 40.1

Licence: CC BY-NC-SA

Creator: Yanina Bellini Saibene

Source: Author's own

Notes: print only



Figure 40.2

Licence: CC BY-NC-SA

Creator: Yanina Bellini Saibene

Source: Author's own

Notes: online only

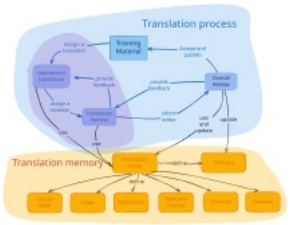


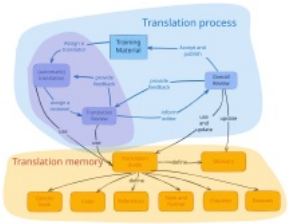
Figure 40.2

Licence: CC BY-NC-SA

Creator: Yanina Bellini Saibene

Source: Author's own

Notes: print only





This book is a collection of 36 articles written by 75 educators who teach programming in a wide range of contexts.

Our stories focus on teaching inclusively with respect and curiosity, and learning with joy, openness, kindness, and community spirit.

We are a grassroots teaching-focused community of practice. Our students learn coding superpowers in psychology, music, medicine, business, art, maths, and many more disciplines.

We understand coding as a way of thinking, problem solving, and empathising; not just typing words and brackets.

But since many of our students did not learn to code previously, together we conquer their own (and society's) misconceptions about what code is, and who gets to become a great programmer. (spoiler: they all do!)

This book collects stories, experiences, and insights from our teaching; we hope that these will inspire you and give you new ideas to incorporate into your own practice.

Co-written by 75 authors.

Edited by Brittany Blankinship, Pawel Orzechowski, Charlotte Desvages, Kasia Banas, Umberto Noè, Chris Oldnall, Serveh Sharifi Far, Clare Llewellyn-MacRae, Beatrice Alex, Ozan Evkaya and Franziska McManus.

Project coordinated by Franziska McManus.

Written by Humans
Free to Access - Free to Share
teaching-programming.github.io/book