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.