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/

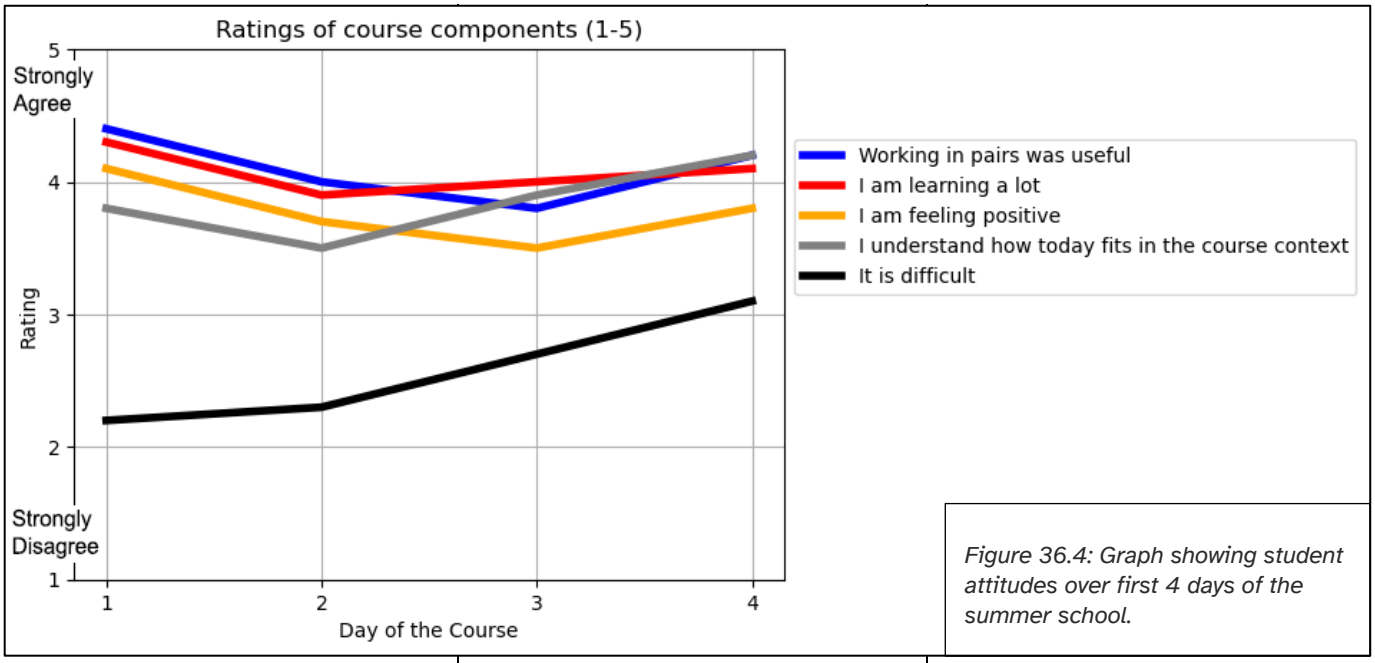


Figure 36.4: Graph showing student attitudes over first 4 days of the summer school.

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.*

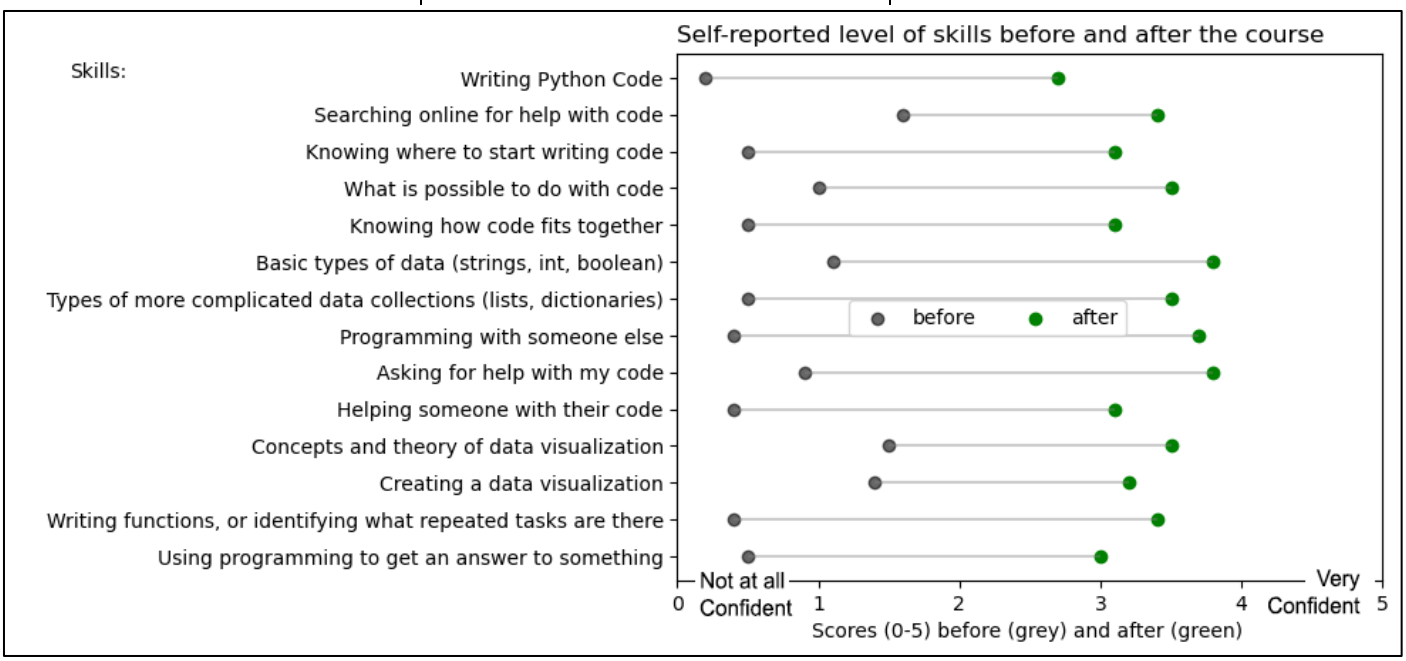
Have you ever tried writing a "gratitude journal" at the end of the day, by listing things that you are grateful for? It consolidates experiences you had, frames them in positive ways and enables incorporating them into our fabric of experience (Bono et al. 2020). To reflect on the personal and transformative 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

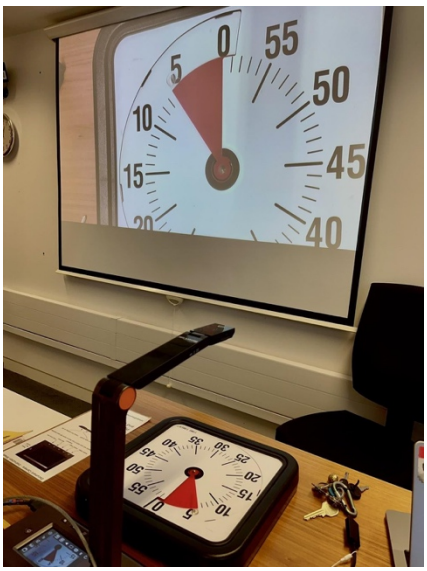


Figure 36.8: 'Time Timer' is a large clock designed for primary schools which we used to indicate when driver and navigator should switch roles.

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).

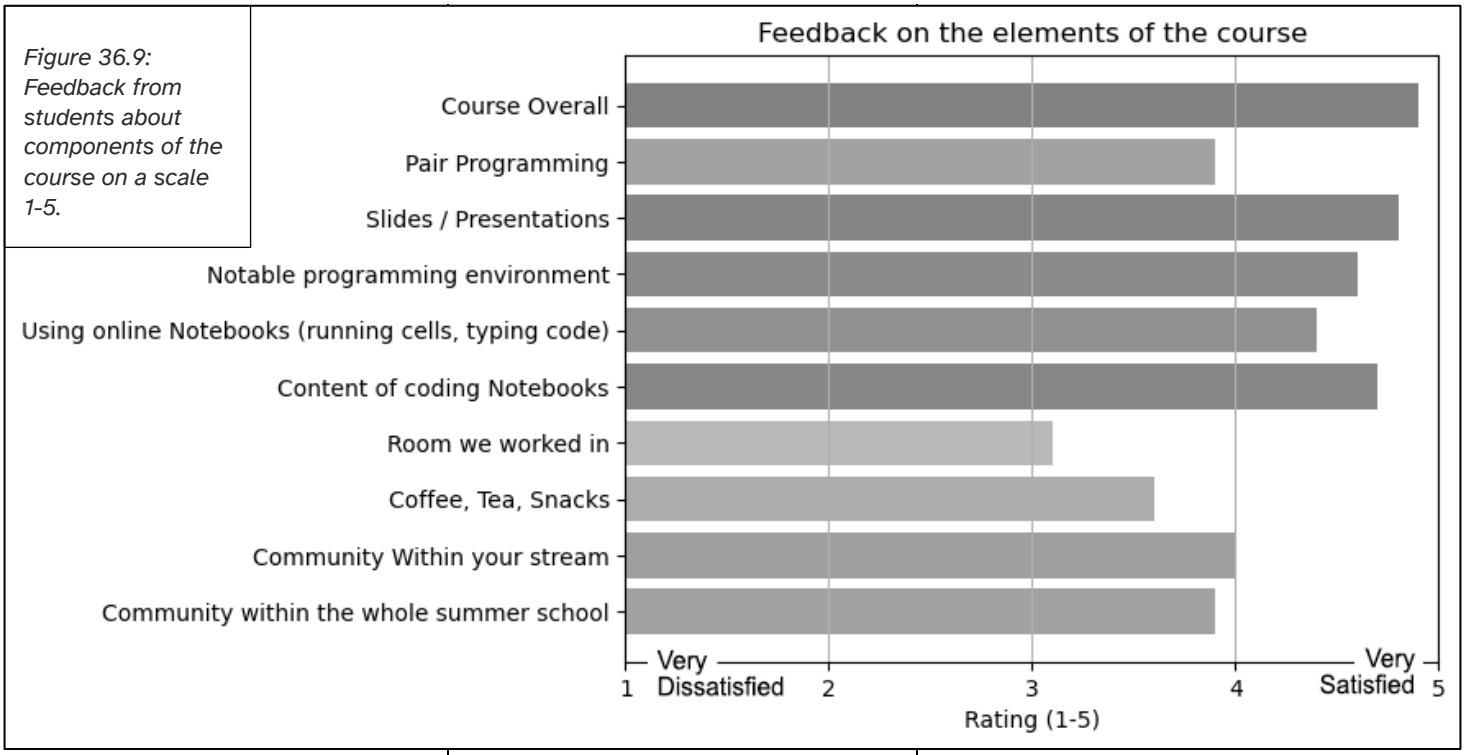


Figure 36.9: Feedback from students about components of the course on a scale 1-5.

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.