Teaching Programming across Disciplines

Table of contents

1 Teaching Programming across Disciplines **4**

2 Title of Example Chapter (as will apper on top h1 header and in table of contents) **5**
2.1 First: things to include 5
2.1.1 Subsection references 5
2.1.2 Subsection link types 5
2.1.3 Subsection images 6

3 Sequential vs. Simultaneous: Approaches to Learning Programming and Statistics **8**
3.1 Authors: Robert S. Young, Rebecca L. Colquhoun, Tiago A. Marques, Brittany Blankinship, Ozan Evkaya 8

4 Introduction **9**

5 Teaching statistics before programming **10**
5.0.1 Example: 10

6 Teaching programming before statistics **11**
6.0.1 Example: 11

7 Teaching programming and statistics together **12**
7.0.1 Example: 13

8 Practical strategies for teaching statistics and programming **14**
8.0.1 Example: 15

9 Conclusion **16**

10 Overcoming coding anxiety: learnings and strategies **17**
10.1 Authors: Tiago A. Marques, William P. Kay[1], Chris Oldnall[1], Chris Sutherland[1], Robert S. Young[1] 17
10.2 Recognising and Refocusing Anxiety 18
10.3 Building Confidence Through Active Learning 20
10.4 Support and Collaboration Beyond the Classroom 22
10.5 Key Takeaways 24

11 Structured group work with assigned asymmetrical roles and switching - Lessons from Pair Programming across disciplines **25**
11.1 Authors: Charlotte Desvages, Pawel Orzechowski, Brittany Blankinship, Umberto Noè 25
11.2 What is "pair programming"? (and why should you care?) 25
11.3 Structured roles for active learning 27
11.4 Fostering peer learning and community 30
11.5 Tackling issues as a community of practice 32
11.6 This blog post was pair-programmed 33

Bibliography **34**

1 Teaching Programming across Disciplines

Welcome to our book!

main page? preface? title etc

2 Title of Example Chapter (as will apper on top h1 header and in table of contents)

2.1 First: things to include

2.1.1 Subsection references

An example citation (Williams 2010).

An example citation Williams (2010).

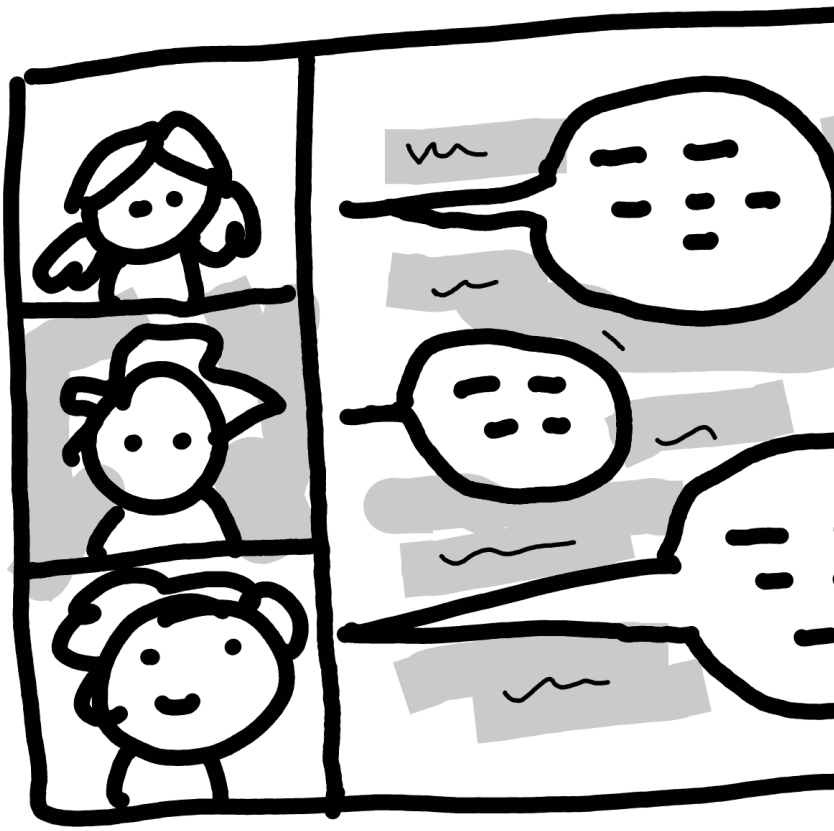
Inner book citation: (Sharifi, Qu, and King 2026)

2.1.2 Subsection link types

https://quarto.org

example link

2.1.3 Subsection images



picture: with caption, source, title, alt and link

picture wrapped in a link, this way picture links to somewhere

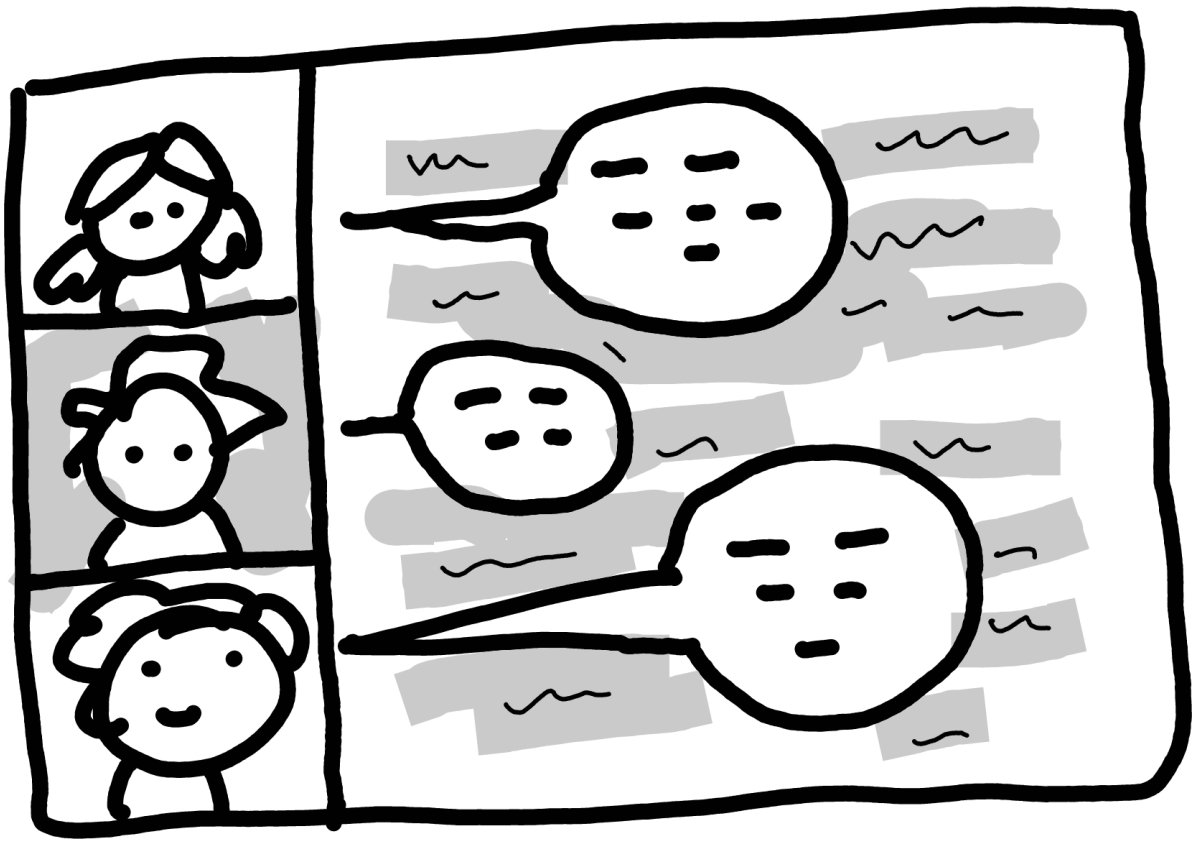


Figure 2.1: Some Caption

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

3.1 Authors: Robert S. Young, Rebecca L. Colquhoun, Tiago A. Marques, Brittany Blankinship, Ozan Evkaya

4 Introduction

We live in a data driven world. Much research in STEM departments requires skills in data description, prediction and inference which are usually developed during statistics modules but, increasingly, non-trivial statistical analysis requires some degree of computer programming. Therefore, for many undergraduate degree programmes, two closely interrelated but fundamentally distinct disciplines, programming and statistics, must often both be taught to undergraduates. There are different possible approaches regarding how to teach 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. Based on our own teaching experiences, we discuss the advantages and disadvantages to different approaches in an attempt to provide guidance on the best way to approach curriculum design.

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. 2022). How to do so while acknowledging that programming is also an important and related skill has received less focus which we hope to begin to address here. In all scenarios teachers need to be mindful of their own educational context, the learning outcome of their course and, above all, what knowledge and skills they want students to get out of learning both programming and statistics.

5 Teaching statistics before programming

Based on our experience, this is perhaps the original or "old school" approach. Many authors in fact were taught in such a way in their own undergraduate degrees prior to the advent of high quality open-source programming languages (e.g., Python, R, etc.) being used widely in academia. The primary advantage of this approach is the focus on core theoretical concepts in statistics (e.g., probability distributions, critical value tables) without additional considerations regarding technology or implementation. In disciplines where students are likely to need to perform statistical analyses in different applications and using a range of technology depending on the subfield they study, this may be the most useful approach. This is also true in disciplines where computers are not always available such as fieldwork, where we want students to be able to identify where they could do 'back-of-the-envelope' statistical analyses. This approach serves to more prominently highlight to students that statistics need not always require computational implementation, which can be lost when programming and statistics teaching are commingled. However, from our experience and perspective, this may be less preferred in undergraduate contexts where the primary subject area is not statistics or programming as this can lead to the links between statistics and programming being lost - potentially leading to a reduced insight into the applications of both.

5.0.1 Example:

A first year geoscience course, which first teaches statistics independently of programming before moving on to a combined approach using the Python programming language. Later in the year, students were on fieldwork at the "Ammonite Pavement" in Dorset, and were asked to measure the size of some ammonite fossils. As they were comfortable with pen-and-paper statistical analysis, they took the initiative to more rigorously find the distribution of ammonite sizes, and to calculate how many ammonites they needed to measure to be confident of having a representative sample. Though the statistics course did continue on to students applying programming to solve statistical questions, the in-the-field experiment was made possible by students first learning to do statistics by hand.

6 Teaching programming before statistics

A complementary approach we have identified is to first teach programming, and then follow this with statistics lessons that make use of students' programming skills. Here, students have time to consolidate their programming skills and build their programming mindset whilst still being able to build knowledge and understanding through a computational investigation of statistical concepts.

The main disadvantage of this approach is that it requires more curriculum time, which many programmes cannot afford. It also requires that the two courses are designed with interaction in mind. For example, using the same programming language in both courses (e.g., jamovi), or purposefully keeping the programming course language agnostic (e.g., giving examples in multiple languages). Teaching statistics in a manner which is reliant on the use of programming may present a barrier for students who are less confident in programming. Indeed depending on the overall curriculum design, students may have forgotten much of the programming learnt by the time they are learning statistics.

6.0.1 Example:

The concept of variance was introduced using a programming approach prior to describing the formal statistical method when teaching Analysis of Variance (ANOVA) to a second-year undergraduate course in biomedical informatics. Students were first shown how to use simulations (sampling within replacement) to compare differences between pairs of samples within and across treatment groups. These empirical results were used to derive the probability of these groups being statistically different from each other and introduced the concept of variance as a quantitative measure. In the subsequent week, the formal mathematics behind the ANOVA were introduced. Students learnt how to implement this using R and, when reasonable, to perform post-hoc tests to determine which treatment group pairs were significantly different from each other. This deliberate approach allowed students to fully comprehend the underlying theory before learning how to implement this and interpret the output in real-world scenarios.

7 Teaching programming and statistics together

The final approach we have identified is to teach programming and statistics together. This can take multiple forms: within a single module one can teach either programming or statistics first, and then immediately use these skills to teach the other, or the teaching of programming and statistics can be truly interleaved.

Teaching these subjects together has clear advantages. Students can experience a more immediate and deeper understanding of statistics, since they can independently probe the concepts being taught, e.g by running simulations. This could facilitate students "building" their statistical knowledge, after an introduction to a new programming and statistics concept each week. For those students where the barrier to statistics learning is past experience with maths (Pletzer et al. 2010), presenting statistical concepts through code snippets may reduce this barrier compared to presenting the same idea using mathematical notation. However, for others the opposite may be true and they could find it easier to explore concepts programmatically after a formal mathematical introduction.

A major disadvantage of combining the teaching of programming and statistics is the risk of cognitive overload. Where both topics are taught concurrently, students do not have the time to consolidate information into their long-term memory and thus the working-memory load is higher (Sweller 2018). Many students will experience hurdles in their learning of both statistics and programming, and these may interact to further impede student attainment compared to students who are more comfortable with one or both topics. For students who are not being taught in their native language, the need to acquire domain-specific vocabulary in two topics at once may further exacerbate issues of cognitive overload. Thoughtful course design can reduce this burden, but it is likely to remain higher than with standalone courses. Besides, the volume of the weekly content should be balanced and tied sequentially in a logical flow to avoid further issues on students' cognitive overload. There may also be concern that students will experience reduced development of computational and algorithmic thinking, since they may only use programming to do statistics. For some disciplines, this may be an appropriate outcome, if the field generally only uses programming for statistical applications and analysis. However, where the course aims to set students up to progress to higher-level programming courses, students may benefit from programming teaching being focussed on a broader range of applications. Besides, the use of suitable programming language can be a key aspect that may impact the student learning performances positively or negatively.

7.0.1 Example:

After seeing a concept such as linear modeling as a part of statistical modeling toolkit, 1st-year undergraduate students from different mathematical degree programmes were shown the related code snippet. Slides either showed the mathematical and code representation side by side or sequentially within this same class, allowing students to make connections across the two approaches.

After lectures, students participated in workshops where context-specific questions with new data sets were shared to give students a fresh start each time. For workshop exercises, they needed to either adapt code snippets based on the given data or create new versions of code to solve the given exercises. It is important to leave sufficient pauses - either during or between classes - during this style of delivery to allow students to fully digest the content from both disciplines.

8 Practical strategies for teaching statistics and programming

We next include a number of specific approaches to teaching both programming and statistics. Firstly, we believe that it is important to embed all teaching of these topics into the subject matter of the overreaching course or programming, particularly when programming or statistics are not the principal teaching focus. This may include creating appealing visualisations, testing data to determine whether the patterns observed are statistically significant, or discussing the conclusions that can be obtained in the research field from this work.

Secondly, statistical methodologies should be described clearly using a computational approach. While it is helpful to discuss the mathematical approach to generating a test statistic, we recommend that this is routinely followed up with the programming approach required to generate the same result. In this way, the close relationship between these two types of work will be demonstrated while also consolidating the underlying statistical concepts by repeating them from different perspectives.

A third strategy is to develop a "programming or coding mindset" by foregrounding the use of programming as an important transferable skill in itself, rather than solely a tool to support the work of performing statistical analysis. Students need to be taught how to interact with data using programming languages (e.g., R and/or Python) rather than directly through apps (e.g., give careful consideration to file paths, focus on good naming conventions in code, not using spaces in file names, and discriminating characters, numbers and special characters). Time should be devoted to debugging code, perhaps through live coding during lectures.

Finally we note that it is critical to motivate and enthuse students about the potential for both statistics and programming. For example, using a computational approach to perform statistical analysis can increase the reproducibility of the obtained results and improve the confidence that genuine scientific discoveries are reported. These skills are also highly in demand in the job market. As an illustrative summary, the gap in data-driven technical skills reported in 2023 in the UK includes specialist data skills (including statistics), programming and software engineering skills to manage and analyse data sets (Fearns, Harriss, and Lally 2023).

8.0.1 Example:

In teaching a postgraduate course that covers both Python and R, a course activity was introduced to promote the development of a "coding mindset" called 'Error of the Week'. This feature celebrates errors as a learning opportunity rather than a source of embarrassment or frustration. Students share errors on discussion boards, including solutions if applicable. The course team responds to every post and during the live session the course lead or tutor highlights a particularly interesting or common error to the class, debugging it live. These discussion boards then act as a common error glossary for the course. These strategies reduce the hurdle that programming can often present to students, allowing for more time to focus on understanding and interpreting statistical outputs.

9 Conclusion

We have outlined three approaches for teaching programming and statistics: teaching statistics first; teaching programming first; and teaching both skills together. These approaches each have advantages and disadvantages, 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 for students, which will vary depending on the educational context and prior student experiences. Considered design of teaching activities can overcome many of the barriers that students may experience through their programming and statistics learning. We hope that through highlighting the advantages and disadvantages of different approaches this chapter will help educators make strategic decisions about how best to teach and encourage the learning of programming and statistics to their students.

Links to other chapters

Possible links to other chapters, which we might need to articulate with (this might not be an exhaustive list, just those we identified so far - might be useful info for the editorial team too)

1. Curriculum Design Overview
2. A Practical Guide to Teaching Python as a Computational Tool for Introductory Data Analysis
3. A guide to empowering students to develop a coding mindset

10 Overcoming coding anxiety: learnings and strategies

10.1 Authors: Tiago A. Marques, William P. Kay[1], Chris Oldnall[1], Chris Sutherland[1], Robert S. Young[1]

1 Authors contributed equally and are listed alphabetically.

Programming or coding 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). Coding anxiety 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), amongst others. 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). Indeed, computer anxiety could be attributed to, for example, variable exposure to coding and an over-reliance on connected digital ecosystems during childhood (*How's Life for Children in the Digital Age?* 2025). The attribution of coding anxiety, however, is a more nuanced issue. Different levels of coding anxiety for mathematics and biomedical undergraduate students exist (Miller and Pyper 2024), with the former finding coding to be more enjoyable, and hence having lower anxiety levels, than the latter. 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 that coding anxiety can be attributed to a variety of sources (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 tool encountered when taking classes involving statistics, data analysis, experimental design, and modelling. It is possible that these students have had no prior exposure to coding, and their initial interactions will define their longer-term relationship with coding, and as a consequence, how they navigate an increasingly data-centric world where coding is an attractive, liberating, and increasingly essential skill.

10.2 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 one to avoid future associated triggers in an attempt to feel some short-term relief (Hofmann and Hay 2018). However, such avoidance can lead to heightened anxiety in the long term (Mkrtchian et al. 2017). Since the brain learns that avoidance temporarily decreases anxiety, even if only momentarily, it may eventually prevent an individual from engaging 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 **TODO: missing reference : Kampmannetal2019 ?** .

We believe that recognising learners' coding anxieties upfront is an important first step towards demystifying it. 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. Cartoons may also be an effective way of making anxiety itself less daunting (Figure 1). If existing online content is not enough, and making cartoons is not one's gift, AI-generated images could easily be created for that purpose - just make sure people have 5 fingers!

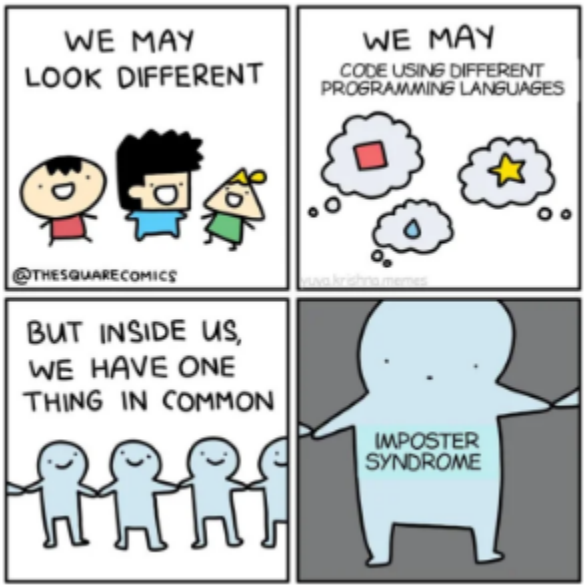


Figure 10.1: Figure 1 - An example of a humorous cartoon that normalises coding anxiety (https://www.thesquarecomics.com/)

Given anxiety's negative feedback loop, the best way to combat anxiety might be to break

this vicious cycle (Figure 2). As an instructor or teacher, this can be achieved by providing positive experiences with coding. Importantly, after acknowledging that coding anxiety is real, the solutions should be presented along with how, as a teacher, overcoming their anxiety will be supported. For example, reflecting on their personal experience, the teacher could describe how they became at ease with coding. Such a role model could serve as a powerful catalyst for students to realise they can overcome their own coding anxiety too.

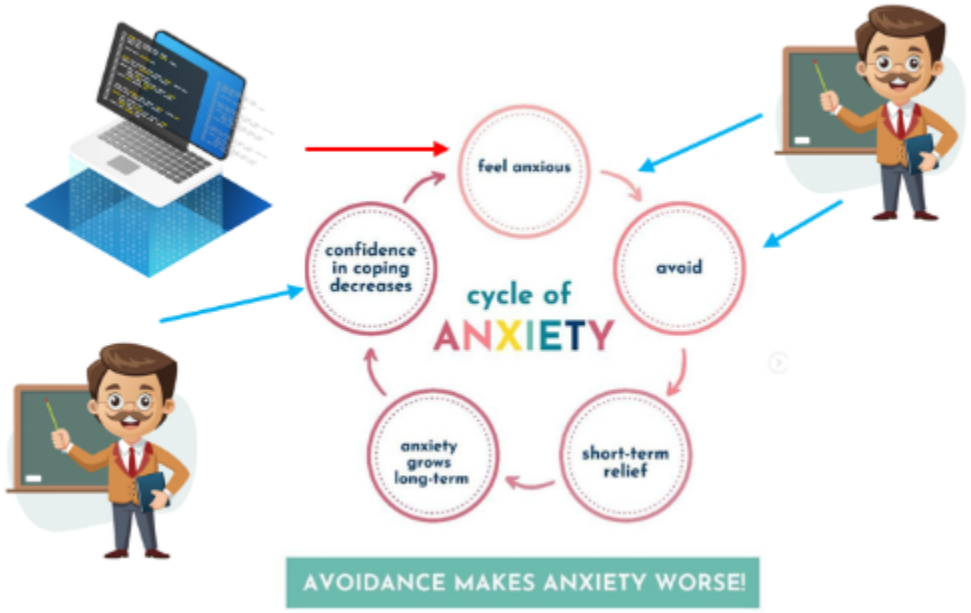


Figure 10.2: Figure 2. The cycle of anxiety. Coding is identified as the trigger and the teacher serves as a diffuser to intervene in multiple places and in multiple ways to break the cycle. Adapted from https://www.mentalyc.com/

Gauging the class's perception towards coding should prove useful in lowering overall anxiety. Collecting information about a priori fears or causes of anxiety related to one's course, including - but not exclusively - coding anxiety, as well as course expectations, provides clues to what might be good strategies to adopt. In-class feedback tools like Mentimenter (www.mentimenter.com), allow students to provide feedback, anonymously and in real time, to make the course more accessible in an informal, friendly, and interactive way. This could 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.

10.3 Building Confidence Through Active Learning

Early hands-on activities involving code, but not necessarily creating their own code, could prove beneficial by focusing on the end product rather than the means to get there. 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 to do 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 (Figure 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.

TODO: missing image? please keep in mind naming convention

Fig. 3. Example of a beautiful visualisation created using R (a map of elephant movements through Kruger National Park). Produced by Dr Pratik Gupte (Imperial College London) and reused here with his permission.

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 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, 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. 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.

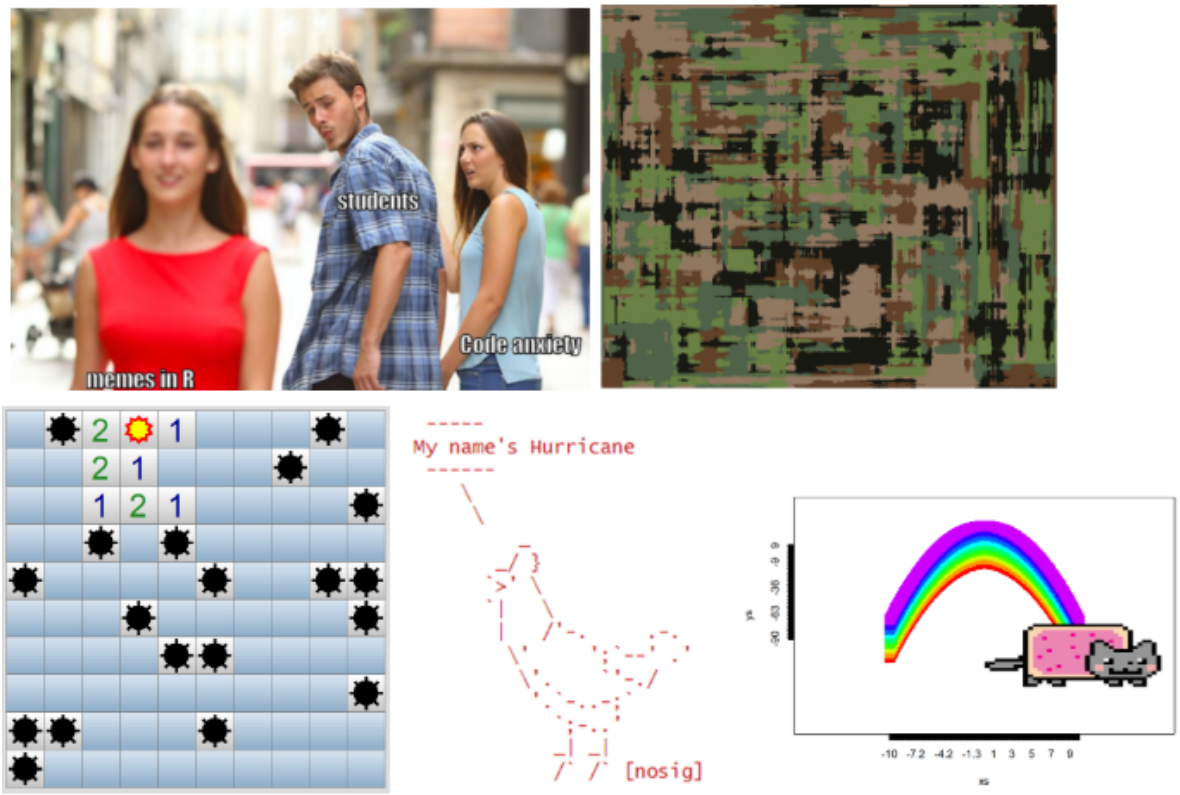


Figure 4. Examples of how R coding can be fun. Top-left: A meme showcasing how memes can reduce code anxiety; this meme can be recreated with 2 lines of code using package "memer". Top-right: A painting created in "aRtsy", a generative art R package. Bottom-left: An instance of a minesweeper game created in package "fun", demonstrating the authors inability to play it. Bottom-centre: An example of an animal that can be drawn in R syntax using the "cowsay" package. Bottom-right: The "CatterPlots" package can be used to generate scatterplots that include cats.

Live coding can be one of the most effective tools to demystify programming and reduce coding anxiety. When teachers write and run 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, eg, 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 a tutor 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 (link to chapter Sequential vs. Simultaneous: Approaches to Learning Programming and Statistics). Common wisdom also says that the first step is the hardest for any journey. In our experience, if students start with small, manageable, and clearly described tasks, their initial experience with coding will be positive, which will decrease anxiety levels when facing harder tasks. Therefore, we suggest that students are first given easy-to-follow tasks with step-by-step instructions, allowing confidence to grow. 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 newfound confidence in their coding abilities at their own pace.

10.4 Support and Collaboration Beyond the Classroom

Coding confidence often grows most effectively outside formal teaching time. Low-pressure practice, whether collaborative or individual, helps students reinforce skills and approach coding with curiosity rather than fear. Teachers can support this 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 additional materials, can help overcome this. Providing a list of such resources will help students realise that they can learn on their own and that it will give them a head start. In our experience, a particularly interesting resource comes via the R package `swirl` (https://swirlstats.com/). 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' (link to generative AI chapter). 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 (Figure 5 + link to other chapters). Pair-programming might be a useful strategy to lower coding anxiety levels **TODO missing REF Brougham et al. 2020**. Nonetheless, these are not universal findings; see (Krizsan and Lambic 2024) for a case where performance was apparently improved, but without a decrease in anxiety levels. Online pair-programming might be less intimidating for some students, 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. It could be used to the same effect as 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. Nonetheless, 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). **TODO: is this meant to be the same reference?** Regardless of the pair-programming mode considered, we believe that if well thought out and prepared, pair-programing activities can be used to lower coding anxiety levels in a range of students.

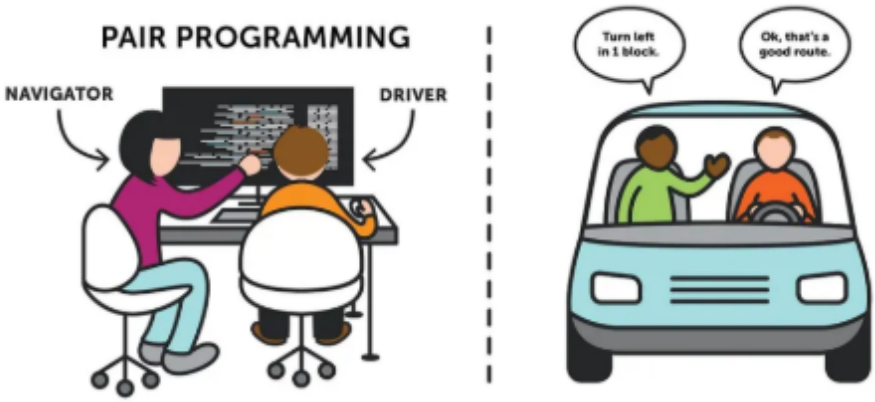


Figure 10.3: Figure 5. Pair-programming can help lower one's coding anxiety.

Students can be encouraged to further their learning journey by 'teaming up' with peers to continue to pair-programme outside of the classroom environment. It has been observed that this can assist with 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, supporting learners beyond the classroom is about 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.

10.5 Key Takeaways

Coding anxiety is real, especially for students for whom coding is a core part of their disciplinary focus. This barrier should be recognised, and we have a responsibility and 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, will support. As a teacher, effective methods for breaking the anxiety cycle are as pedagogically important as content creation. Moreover, you will become an anxiety diffuser, starting by exploring ways to provide rewards from coding activities. In particular, activities allowing students to see the potential 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 strategies discussed above will be fundamental for reducing coding anxiety and improving learning.

11 Structured group work with assigned asymmetrical roles and switching - Lessons from Pair Programming across disciplines

11.1 Authors: Charlotte Desvages, Pawel Orzechowski, Brittany Blankinship, Umberto Noè

In this chapter we will share our experiences with a structured group work format that we believe fosters student engagement, well-being, and transferable skills.

11.2 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.

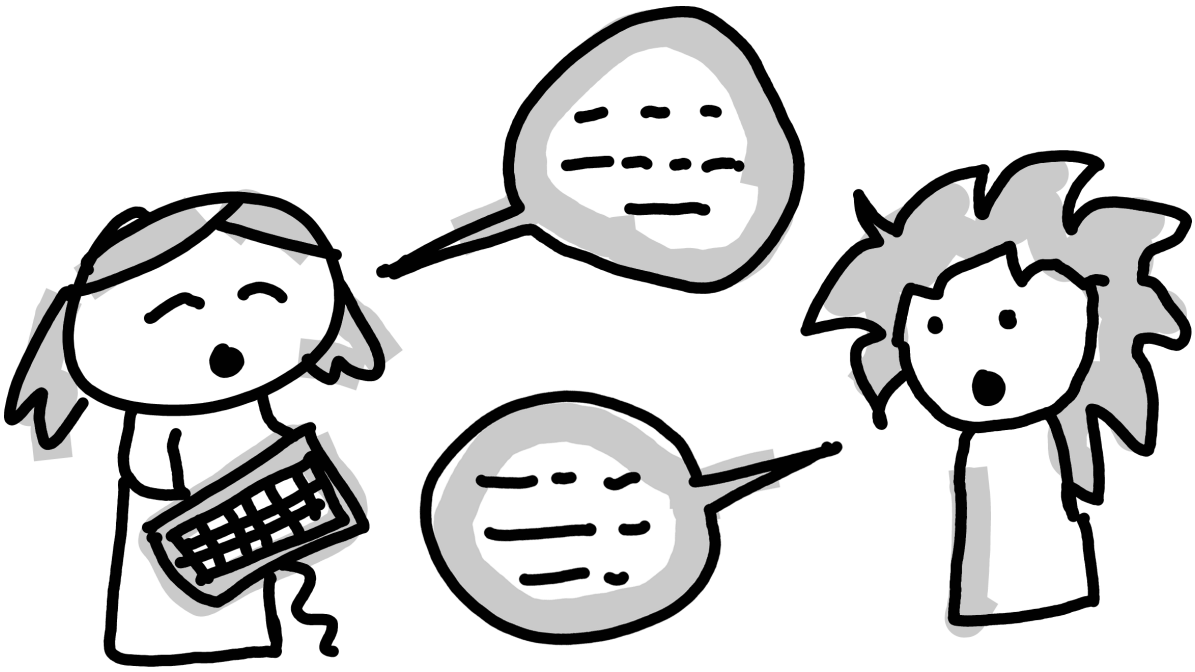


Figure 11.1: "I think your idea will work, let me type it in!"," I see a bracket missing on line 7", "Let's run this code and see what happens.", "Yay, we did it! It works!"

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 (UG and PGT); 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.

11.3 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." - UG, Psychology

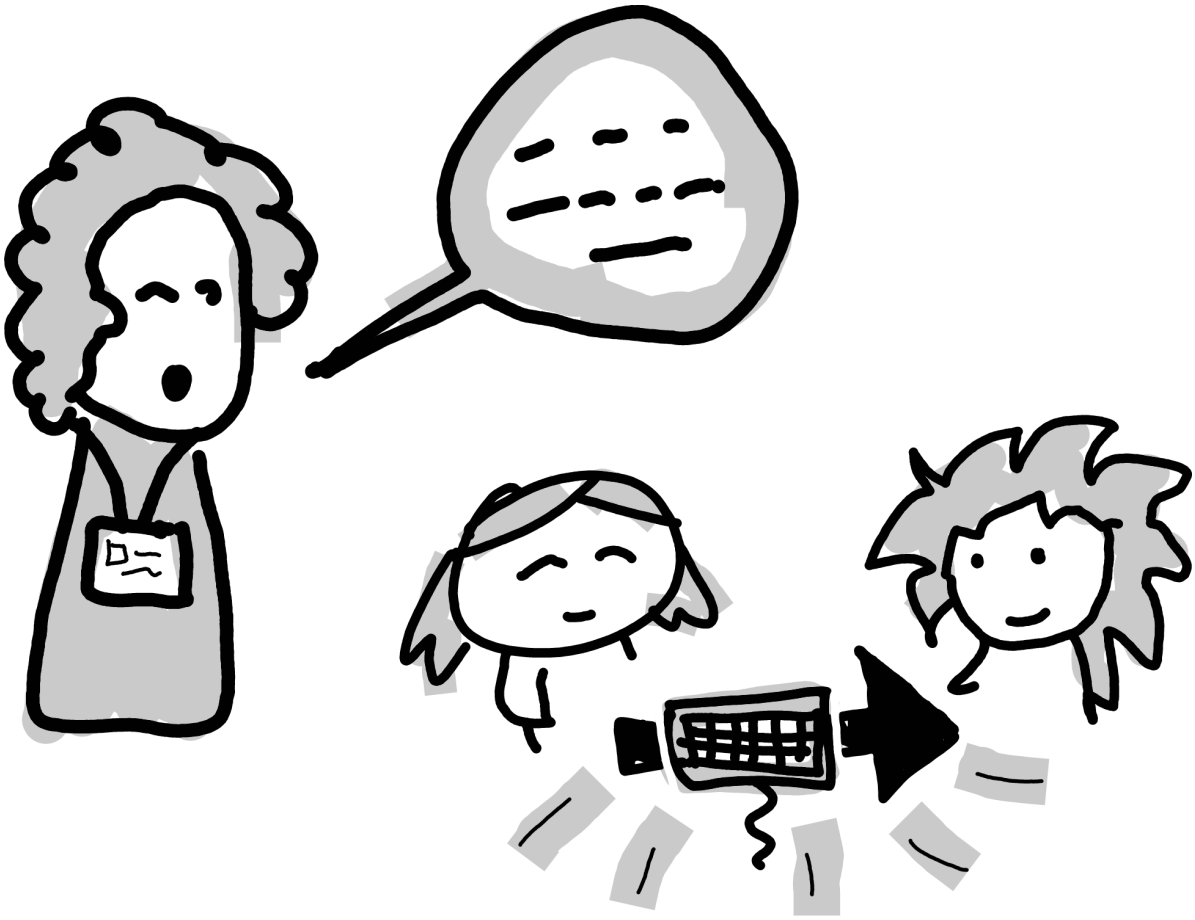


Figure 11.2: “Hello! Which of you is driving now?”, “Is it time to switch drivers?”, “Do you need help getting unblocked or are you just cracking on?”

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” - PGT, 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." - UG, 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.

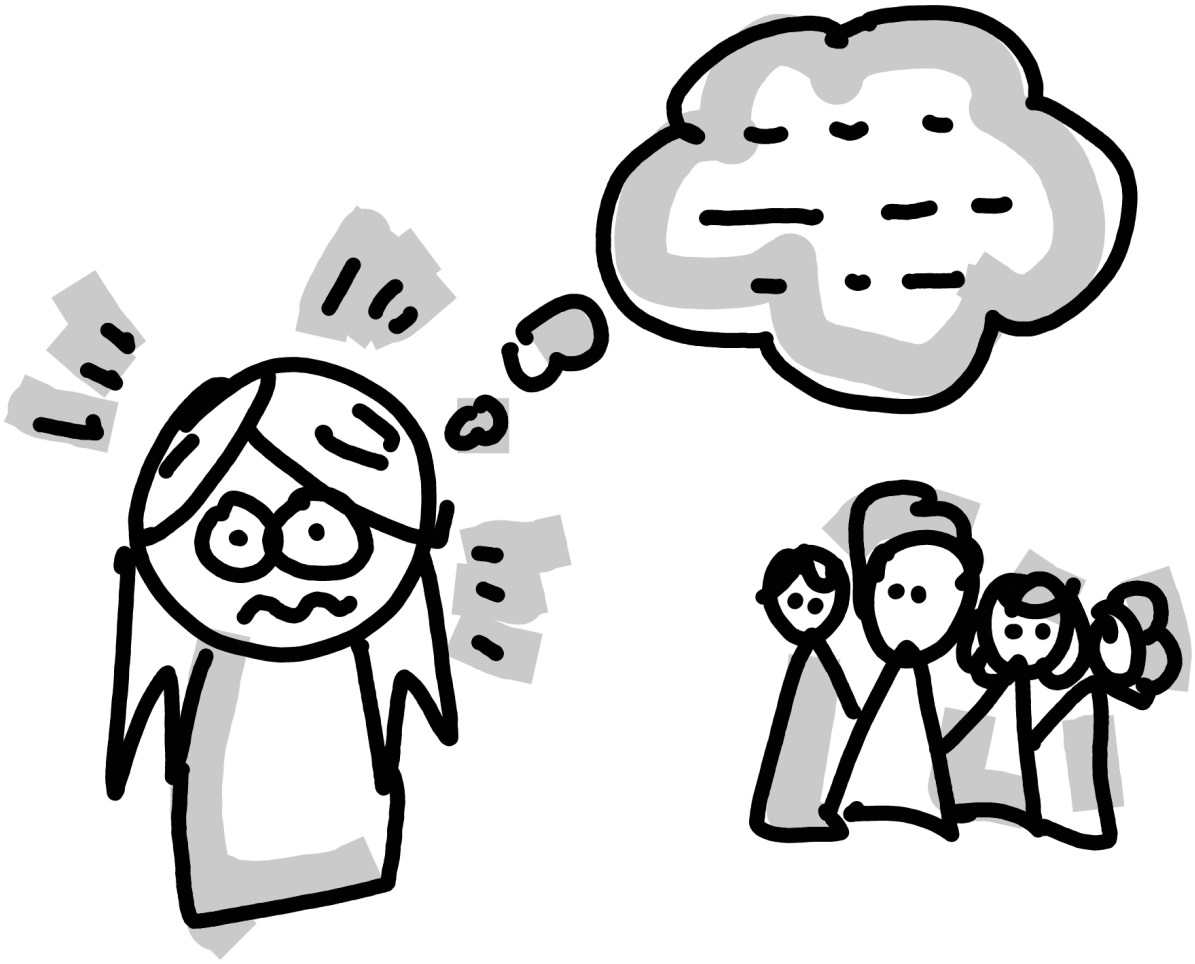


Figure 11.3: "What if I end up in a pair with someone who is mean, or much more advanced than me?", "I'm uncomfortable in socially complex situations. Will I know what to say and do in a group…", "Should I just always work with the same friend, or would I miss out on meeting new people, and learning 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.

11.4 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." - PGT, 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." - PGT, 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." - UG, Psychology

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

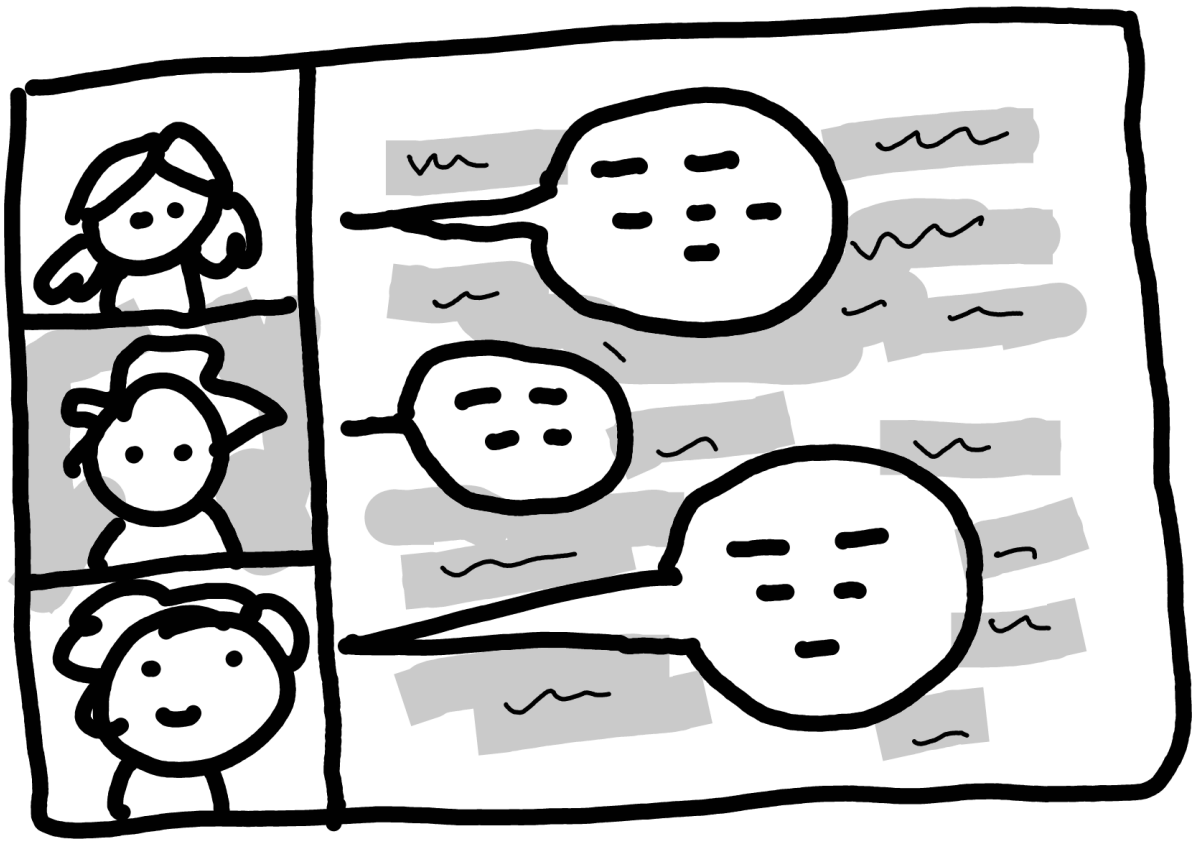


Figure 11.4: “So the way I understand it, is …”, “I see another way to solve this!”, “That’s cool! I never thought about it this way”

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.” - PGT, 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." - UG, Psychology

> "[It] was nice to get some 1-1 interaction through the pair programming in what could have been an otherwise solitary course." - PGT, 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." - UG, 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." - PGT, Medical School

11.5 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." - UG, 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.

11.6 This blog post 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 blog post 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.

Bibliography

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.

Bromage, Adrian, Sarah Pierce, Tom Reader, and Lindsey Compton. 2022. "Teaching Statistics to Non-Specialists: Challenges and Strategies for Success." *Journal of Further and Higher Education* 46 (1): 46–61.

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.

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.

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.

Fearns, Josh, Lydia Harriss, and Clare Lally. 2023. "Data Science Skills in the UK Workforce."

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.

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.

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*, 1–7. WCCCE '23. ACM. https://doi.org/10.1145/3593342.3593351.

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.

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.

How's Life for Children in the Digital Age? 2025. OECD Publishing. https://doi.org/10.1787/0854b900-en.

Kaur, Tarandeep, and Samantha Newell. 2024. "The Silent Struggle: Experiences of Non-Native English-Speaking Psychology Students." *Australian Journal of Psychology* 76 (1): 2360983.

Kelly, Martyn. 1992. "Teaching Statistics to Biologists." *Journal of Biological Education* 26 (3): 200–203.

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* 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.

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.

Miller, Ainsley, and Kate Pyper. 2024. "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.

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.

Mustafa, R Yilmaz. 1996. "The Challenge of Teaching Statistics to Non-Specialists." *Journal of Statistics Education* 4 (1).

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.

O'Hara, Robert B. 2016. "On Teaching Ecologists Contemporary Methods in Statistics." Wiley Online Library.

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.

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.

Saltz, Jeffrey, and Robert Heckman. 2020. "Using Structured Pair Activities in a Distributed Online Breakout Room." *Online Learning* 24 (1): 227–44.

Sharifi, Serveh, Ruini Qu, and Stuart King. 2026. "Teaching Programming Across Disciplines." In.

Sweller, John. 2018. "Instructional Design." In *Encyclopedia of Evolutionary Psychological Science*, 1–5. Springer.

Weinberger, Armin. 2011. "Principles of Transactive Computer-Supported Collaboration Scripts." *Nordic Journal of Digital Literacy* 6 (3): 189–202.

Williams, Laurie A. 2010. "Pair Programming." *Encyclopedia of Software Engineering* 2.

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 (January). https://doi.org/10.3389/fpsyt.2021.777190.

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.