Teaching Programming across Disciplines

Table of contents

1 Teaching Programming across Disciplines

Welcome to our book!

We are a community of teachers and educators of computer programming outside of traditional computer science settings. Our 300+ members around the world meet every year for a Winter School[1] and co-author a book called Teaching Programming across Disciplines[2] (this very book).

Our book is an edited, open-access volume comprised of many short chapters written by groups of authors.

We will be launching the **first edition of Teaching Programming Across Disciplines** at the "Summer-time Winter School" on **22 June 2026**!

The **hard deadline for chapter submissions** is **1 April 2026** – meaning your chapter should be edited, polished, and on the GitHub Repository (watch Pawel's video[3] for instructions on how to do this). The book will be growing from now until 1 April 2026 as chapters are added. There will be a print copy available (eventually… stay tuned).

We hold monthly writing retreats, which authors or anyone interested in having dedicated writing time with a group of lovely people are welcome to join. We also have a newsletter with book updates and (interesting) events. If you would like to join our mailing list, email pairprogramming@ed.ac.uk[4]

The Book Team (Brittany Blankinship, Franziska McManus, Pawel Orzechowski, Kasia Banas, Charlotte Desvages, Beatrice Alex, Umberto Noé, Ozan Evkaya, Serveh Sharifi Far, Clare Llewellyn)

[1] https://pairprogramming.ed.ac.uk/category/winter-school/
[2] https://pairprogramming.ed.ac.uk/join-book/
[3] https://media.ed.ac.uk/media//1_7nk68kt4
[4] mailto:pairprogramming@ed.ac.uk

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

2.1 Your first top-level section

2.1.1 A subsection

This is an example subsection, starting with `###`.

2.2 Quarto Markdown quick examples

2.2.1 Citations

An example citation (Williams 2010). You can also cite Williams (2010) in the text, without the brackets.

This is a reference to a chapter of a book (Sharifi et al. 2026). You can find it in `references.bib`; it uses the `@inbook` type.

The "References" section will be populated automatically at the bottom.

2.2.2 Footnotes

This is some text with a footnote[1].

[1] Write the content of your footnote here.

If you need a multiline footnote, indent the following lines with 4 spaces…
…like this. Everything indented will be part of the footnote.

2.2.3 URL links

Use link text[2] to include URL links; please refer to the accessibility guidance in the book submission guidelines.

If it is absolutely necessary to display the full URL in the text, use `<...>`: https://quarto.org

2.2.4 Images

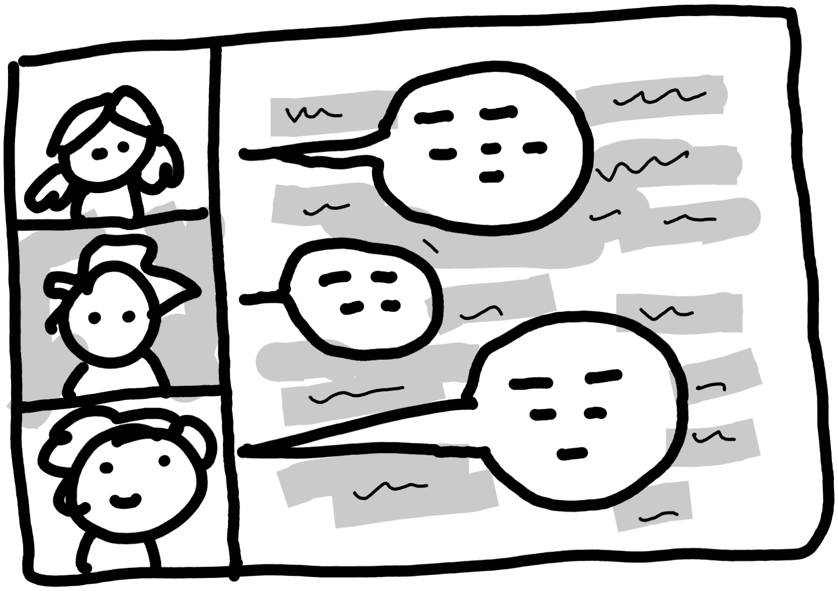

Figure 2.1: This is the caption for an image, sized to take 70% of the screen width.

[2] https://quarto.org

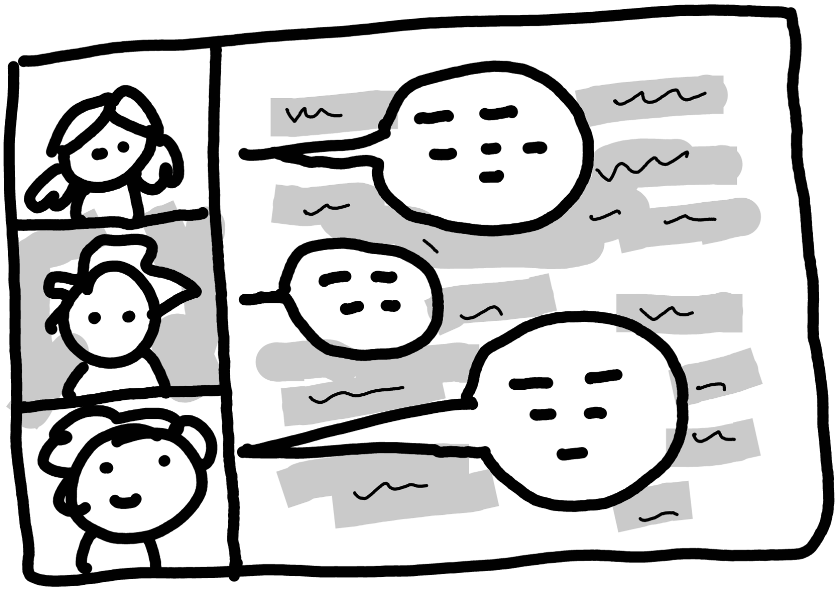

Figure 2.2: This is the caption for an image with a clickable link.

2.2.5 Putting some content in a box

Title of your box

You can put some content in a box like this, called a callout[a], if you want it to be visually distinct from the rest of your chapter. This could be e.g. if you write an example or a short case study.

[a] https://quarto.org/docs/authoring/callouts.html

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

3.1 Teaching Data Analysis and Programming: Does One Need to Come First?

Some introductory data science courses must teach both programming and statistics to students with no prior experience in either. Students may first take an introductory programming course; however, this is not always possible, and often programming must be integrated alongside data science concepts (Colquhoun, Marques, et al. 2026). Teaching a data science or statistics course without a computational tool is not possible due to the amount of computations required in working with even small datasets. On the other hand, syntax-heavy programming courses may not immediately show relevance to data science and often struggle to engage students, who find it difficult to invest in abstract concepts without a clear sense of their practical application.

This chapter presents a structured, stepwise practical guide to introducing Python as a computational tool in an introductory data science course, allowing students to balance learning both technical skills (coding) with conceptual skills (statistical and data science thinking). The emphasis is on six fundamental elements to teach students to complement their data analysis learning in a process that supports a gradual transition from simple scripts to structured analysis. Each element introduces just enough Python to support statistical thinking, promote motivation, and empower inexperienced learners to use this tool for producing meaningful data analysis results.

3.2 Overview of Such a Course

This approach has been implemented by the authors in an introductory data science course for diverse learners at the MSc level in an eleven-week semester (King and Sharifi Far 2025). We suggest structuring such a course so that statistical concepts and methods are taught in lectures, while computer workshop sessions focus on applying the methods using Python. In the workshops, Jupyter notebooks that combine explanatory text and executable code can be used as a common and effective tool to practice coding (Gemayel et al. 2026). This benefits beginners by keeping instructional material and code side by side and reducing the cognitive load that can come with more complex development environments. Furthermore, working on notebooks in the workshops can then be done in the pair-programming format to stimulate student discussion and peer learning (Orzechowski et al. 2026). To support learning, we also recommend integrating continuous assessment throughout the course. Using automated marking tools for coding tasks can provide timely, objective feedback and help students identify specific areas for improvement, which reinforces understanding and encourages regular engagement with the material.

3.3 Core Code Elements

Here we list the core code elements and Python functionalities fundamental to them, and the minimum set of inbuilt or module-specific functions needed to teach them. For each element, we describe its pedagogical goal and value, highlight common coding pitfalls observed in student learning, of which it can be helpful for instructors to be aware, and suggest real-world datasets to use that provide meaningful opportunities for practical application.

3.3.1 Element 1 - Foundational Coding: Expressions, Variables, and Basic Calculations

- **Goal**: Introducing students to how Python solves simple mathematical problems and handles expressions, assigns variables, outputs results, and applies simple built-in functions.

- **Pedagogical value**: This stage invites students to become comfortable with the Python syntax and environment. It allows them to store and reuse results to make their code more readable and flexible. Combining variables with simple functions introduces the core logic of programming. This stage builds confidence by showing how basic Python code can mimic a calculator while adding structure and repeatability.

- **Essential functions**: Creating lists of data using `[]`, assignment operator (`=`), arithmetic operators (`+`, `-`, `*`, `/`, `**`), printing results with `print()`, checking type of variables with `type()`. Some built-in mathematical functions, for example, `round()`, `sum()`, `len()`, `min()`, `max()`, `abs()`.

- **Data suggestion**: Using minimal lists and arrays of numbers and characters is helpful at this stage. Students can be invited to make a small list of their favorite fruits, colors, or a short series of numeric values (e.g., prices, scores).

- **Common pitfalls**: Misspelling variables or functions names; reusing variable names unintentionally; calling an unassigned variable (e.g., the notebook has defined some variables in a cell and students call these variables without executing the cell); forgetting to assign results of calculations to variables; misunderstanding variable type changes (e.g., overwriting a numeric/integer variable with a text/string).

3.3.2 Element 2 - Handling Data with `pandas`: Reading and Cleaning

- **Goal**: Equip students with the essential skills to load, explore, and clean datasets.

- **Pedagogical value**: It is essential for students to learn to work with structured data in the form of data frames. Introducing them to how to load datasets, explore their structure, and select or clean rows and columns is needed for visualisation, modeling, and interpretation.

- **Essential functions**: Importing `pandas`, functions such as `pd.DataFrame`, `pd.read_csv()`, `pd.read_excel()`, `head()`, `info()`, `describe()`, `loc[]`, `iloc[]`, selection of rows and columns with `[]` and operations on them (e.g., `df['col'].mean()`), handling missing data (`np.nan`, `dropna()`, `fillna()`, `replace()`), extra useful functions such as `groupby()`, `filter()`, `apply()`, `value_counts()`, `sort_values()`.

- **Real data suggestion**: Turtle Size dataset (Phillips et al. 2021) is a simply structured and helpful dataset which includes numerical and categorical variables. Penguins dataset (Gorman et al. 2014) is another interesting and manageable dataset with some missing values for students to explore.

- **Common pitfalls**: Confusing `loc[]` with `iloc[]`; forgetting to save cleaned data in a variable; misunderstanding how `[]` behaves differently with rows and columns; forgetting that column names are strings when selecting them (e.g., `df["ColumnName"]`); forgetting to use a list when selecting multiple columns/rows (e.g., `dataframe.iloc[[5,1]]` vs `dataframe.iloc[5,1]`).

3.3.3 Element 3 - Visualising Data with `matplotlib` and `seaborn`: Explore, Compare, and Communicate

- **Goal**: Use visual tools to describe the data, explore distributions, trends, and relationships among variables.

- **Pedagogical value**: Visualisation helps students in conducting exploratory analysis of the data, identifying outliers, checking distributions, and spotting potential relationships between variables, and reinforces the connection between statistical concepts and their visual representations.

- **Essential functions**: Importing `matplotlib.pyplot`, then `plt.plot()` and `plt.show()` with `plt.xlabel()`, `plt.ylabel()`, `plt.title()`. Importing `seaborn` and using the general format of `sns.---plot(data=---, x=---, y=---, kind=---, ...)` for different `kind` of plots.

- **Real data suggestion**: House Sales in King County (Harlfoxem 2016) data is a manageable dataset for students to explore many ways of visualising house prices based on various numerical and categorical features. The Hotel Booking Demand dataset (Antonio et al. 2019) also provides a chance to visualise a hotel room prices and availabilities in Portugal based on several factors.

- **Common pitfalls**: Forgetting to label axes or add titles; confusing `kind` options in plots; overcomplicating plots by including many variables; difficulties with data restructuring if required for a plot; difficulty in understanding the connection between `seaborn` and `matplotlib` for beginners.

3.3.4 Element 4 - Summary Statistics: Describing Data

- **Goal**: Enable students to summarise variables and distributions and examine relationships between variables using descriptive statistics and correlation.

- **Pedagogical value**: Introducing summary statistics measures provides students with the language to describe data precisely. The concept of correlation offers an early opportunity to explore potential associations between variables, before modeling.

- **Essential functions**: Measurements of center and spread of data from `numpy` after importing it. The important functions are, `np.mean()`, `np.median()`, `np.mode()`, `np.std()`, `np.var()`, `np.quantile()`. Calculating Pearson linear correlation between two variables using `corr(method="pearson")`.

- **Real data suggestion**: Sleep in Mammals (Allison and Cicchetti 1976) and Animals Life Expectancy (Che-Castaldo et al. 2019) are interesting datasets about animals with various features in different types that can be used to encourage students to explore the pattern of how long these animals sleep and live.

- **Common pitfalls**: Applying numeric-only summary functions to non-numeric variables; ignoring or not checking for missing values; not inspecting the shape and possible skewness of the distributions.

3.3.5 Element 5 - Normal Linear Regression with statsmodels: Explain and Predict Relationships

- **Goal**: Provide students with tools to build linear regression models to explain relationships between variables and make predictions, using syntax that mirrors traditional statistical notation.

- **Pedagogical value**: Linear models are simple yet foundational, offering students a strong basis for deeper study of statistical modelling. Introducing the normal linear model, along with the binary linear model (logistic regression) if time permits, could be sufficient at this stage. Attention should be given to checking the model assumptions and its goodness-of-fit.

- **Essential functions**: Importing the library `statsmodels.formula.api`, using the `ols` function in the form of `model = smf.ols("y ~ x_1 + x_2", data=df).fit()` and the corresponding required functions, `model.summary()`, `model.predict()`, `model.resid`. Some Useful plots are `sns.lmplot()`, and `sns.residplot()and qqplot()` for visualisation and checking the model assumptions.

- **Real data suggestion**: World Happiness Report (*The World Happiness Report*, n.d.) data from most countries around the world and different years can be used to aim to model the countries level of happiness based on various measured social elements. Programme for International Student Assessment (Organisation for Economic Co-operation and Development, n.d.) data provides standardised exam scores for a large sample of students in different countries along with many social and educational indicators.

- **Common pitfalls**: Difficulty in coding categorical variables; in extending the model formula to variable names including spaces; and in inputting new data in the `predict` function or passing in wrong data types.

3.3.6 Element 6 - Machine Learning with `scikit-learn`: Classification

- **Goal**: Introduce students to basic supervised machine learning for classification problems, using intuitive models like k-Nearest Neighbors, decision trees, and ensemble methods, and reinforcing best practices in model evaluation.

- **Pedagogical value**: Introducing classification methods shows students how algorithms can learn patterns from labelled data to make predictions.

- **Essential functions**: From `scikit-learn` use `train_test_split` to prepare data for training and assessing models. Then apply different classification methods, using `KNeighborsClassifier`, `DecisionTreeClassifier`, `RandomForestClassifier`, `BaggingClassifier`, and `HistGradientBoostingClassifier`, which all need `fit()` and `predict()`. Evaluation of the classification can be done using `classification_report`, `confusion_matrix`, and a simple scatter plot like `sns.scatterplot()` is useful in visualisation.

- **Real data**: Pima Indian Diabetes dataset (Smith et al. 1988) includes various health indicators to use in modelling to predict whether participants would develop diabetes. Behavioral Risk Factor Surveillance System (Centers for Disease Control and Prevention, n.d.) data is available from several years and includes many health indicators and records of presence of many health conditions to use in different classification problems.

- **Common pitfalls**: Not applying the train/test split or mishandling its output; difficulty in choosing and applying the appropriate evaluation methods.

Depending on the course level and learning objectives, Element 6 may be included only if an introduction to machine learning falls within the scope of the course. Here, we presented classification as an example of a supervised learning method, although an unsupervised method such as clustering could also be introduced following a similar structure.

3.4 What Can Students Take Away?

By working through these core elements, students gain a practical foundation in both Python programming and data analysis. This progression helps students move from simple code scripts to structured analytical workflows, equipping them with the confidence and technical ability to engage with real-world datasets. By the end of this material, students should be able to implement core Python functions for data analysis independently and recognise how coding supports statistical thinking. The common pitfalls noted throughout are not mere technical errors, but can be important learning opportunities.

4 Reflections on Online Teaching

4.1 Introduction

In this chapter, educators with a background in teaching online will share some of their experiences. These reflections are intended to be informative and to provide insight, inspiration, and advice to other educators who are undertaking similar teaching. Specifically, the chapter discusses:

- A "Silent Disco" at The University of Edinburgh, where students work through material at their own pace with tutors available to facilitate,
- An introductory Python course developed on Coursera for early-career researchers at Imperial College London,
- Four Massive Open Online Courses (MOOCs) integrated into a Masters of Public Health at Imperial College London, including discussions of assessment and interaction between the learners and staff,
- A selection of blended work-based modules delivered at the University of Strathclyde, designed to be taken by learners while they are in employment.

4.2 Silent Disco

In this section, Lucia describes her experience in developing "Silent Disco" training activities within the context of the Centre for Data, Culture, & Society (CDCS)[1] *at The University of Edinburgh.*

One of the notable online teaching innovations developed during the pandemic as part of the Training Programme was the "Silent Disco." Conceived as a response to widespread "Zoom fatigue" (Sklar 2020), this format deliberately moved away from the standard model of extended video calls. Instead, it sought to create a learning environment that recognised the variety of ways in which participants approach technical training.

At its core, the Silent Disco is a facilitated, partially asynchronous workshop. Participants join a shared Microsoft Teams chat at a scheduled time, and they work through structured materials at their own pace, rather than listening to a live lecture. These may include coding notebooks, short tutorials, or step-by-step guides. Throughout the session, which normally

[1] https://www.cdcs.ed.ac.uk/

ranges between two to three hours, instructors are present in the chat to provide support, answer questions, offer clarifications and point to further learning material. The absence of video and audio calls reduces the cognitive load of constant online presence, while the live facilitation ensures that learners are never left entirely on their own (Park et al. 2025). The possibility of concentrating the focus solely on individual learning experiences was recognised as a particular point of strength by the attendees, as testified by the feedback we received:

> "Silent disco is just an awesome format for learning digital methods, better than live/synchronous webinars because a lot of digital method steps require the learner grappling with the code/steps ourselves. which is better done when alone (in silence)". — Learner feedback

To further support the students, the chat remains open for the duration of the semester following the event. This extended access allows participants to revisit the workshop materials at their own convenience and attempt to apply the tutorials to their own datasets. Moreover, attendees are encouraged to use this platform to share relevant updates, insights, or interesting findings with their peers, fostering a continuing community of learning and collaboration. This ongoing engagement ensures that the Silent Disco remains a dynamic resource beyond the initial workshop session.

The Silent Disco format is particularly well-suited to audiences who may not see themselves as "coders" and have limited time for learning new methods. Based on feedback, mostly from staff, they appreciated the format because they couldn't commit to a full course, but also found complete self-learning challenging. For students and researchers in the arts, humanities, and social sciences, technical content can often feel intimidating, especially when delivered at the pace of a conventional lecture. The Silent Disco allows participants to pause, reread, and experiment without the pressure of keeping up with a live demonstration. At the same time, it provides an immediate channel for assistance, which is often crucial when encountering coding errors or unfamiliar interfaces.

CDCS has applied the Silent Disco approach across a diverse set of workshops. Some have focused on foundational skills, such as database design and SQL querying. Others have introduced intermediate methods for text analysis, sentiment detection, and data visualisation. More creative sessions have included work on explorations of large language models and the usage of regular expressions to correct OCR[2] errors. This range illustrates the flexibility of the Silent Disco model: it accommodates both the practical skills that underpin reproducible research and more experimental engagements with data.

While the Silent Disco format offers a dynamic and flexible approach to learning, it does have certain limitations. For participants who prefer direct, real-time interaction, the absence of live video guidance can make it difficult to deal with complex errors, which could be more efficiently resolved in person or through screen sharing. Additionally, managing multiple asynchronous

[2] Optical Character Recognition (OCR) is a technology used to extract machine-readable text or characters from raster images or scanned documents.

queries can be challenging for facilitators, potentially leading to delays in addressing individual issues.

What the Silent Disco demonstrates, though, is that online teaching can be both efficient and versatile. By shifting the emphasis away from continuous video interaction, it reduces fatigue while also validating different learning styles. For non-coders in particular, it creates a safe space to approach computational methods incrementally, supported but not overwhelmed. Although born of the pandemic, the model continues to offer a valuable pedagogical alternative, reminding us that inclusive design often begins with the recognition that there is no single "right" way to learn.

4.3 "Introduction to Python for Researchers", a Coursera MOOC

In this section, Chris describes his experience developing an introductory Python course on Coursera for the Early Career Researcher Institute (ECRI) at Imperial College London.

Between Summer 2023 and Winter 2024, I developed a course named "Introduction to Python for Researchers" in conjunction with the Interdisciplinary EdTech Lab (IETL)[3] at Imperial College. This course was hosted on Coursera and was based on a previous course I had developed in Blackboard. The course is intended to take 10-15 hours to complete and covers the basics of Python programming. The course is primarily aimed at early career researchers (such as PhD students and postdocs) and is designed to be taken on-demand by learners when they feel it benefits their professional development.

We aimed to distinguish the course from other introductory Python courses by focusing the examples on simplified scientific problems, and best-practice and tools and techniques that would be useful in the real world. In formulating the course, we chose to minimise the number of videos, using images, gifs and embedded runnable code cells in the course pages as much as possible. This is because the time investment to create and maintain videos is much greater than these other media.

The development process took around 18 months where it made up around half of my full-time workload. I developed the content and my colleagues at the IETL formatted it on Coursera, and advised on the pedagogy of online courses. At the time of writing, the course has been live for around 8 months and has had around 1300 enrolments, with 145 completions. We were able to configure the course so that accessing the full version of the course as a member of the public required a paid Coursera subscription, generating some income for us. For students at our university, however, they were able to access an identical duplicate of the course for free using their institution's email address.

Developing this course was hard and time-consuming. The course has 110 pages and contains around 60,000 words. Although the subject was very well-known to me and I had previously

[3] https://www.imperial.ac.uk/interdisciplinary-ed-tech-lab/

produced similar courses to draw inspiration from, creating the content took a long time. There were normal pedagogical considerations relating to what to cover, how to order it, and what examples and exercises to include. There were also issues relating to Coursera and producing a MOOC specifically — how to render and format the content on the site, how to direct students to forums with questions, and how to use the technology of Coursera to facilitate examples and exercises. Resolving these questions also required the most intensive collaboration with the IETL. In all, developing the content probably took around half the effort, with the other half spent on issues relating to Coursera.

Coursera offered some useful tools for creating the course — one of the most useful being the Coursera Lab. This is a configurable containerised environment with a Visual Studio Code interface. We used this for all of the exercises and some of the examples. For the exercises, we were able to define a suite of unit tests that could check the learner's code to see if it behaved properly. By anticipating some common mistakes a student might make, we were able to include specific feedback messages to help learners correct common mistakes. Coursera also allowed for passing these tests to be required for progress of completion of the course; however, we chose not to do this, and to make only passing the multiple-choice exam at the end of the course a requirement.

One ongoing cost of this course is monitoring. As the main tutor, I check the course's forums once a week for questions and problems. At the current modest user base of the course, questions are relatively rare, but if the course got more popular, this would eventually create a significant workload. On one occasion, the way Coursera implemented the Labs behind the scenes unexpectedly changed and this broke the grading in all of our labs. This led to many questions and lots of feedback from struggling learners. I was grateful that my colleagues at the IETL were able to quickly identify and fix the problem, but this was an unexpected and significant amount of work. It seems possible this could occur again.

Overall, internal and external students both seem happy with the course. For our internal students I think there is a slight improvement in experience compared to our old Blackboard course that this course replaced. A downside is that developing the Coursera course did take more time and effort as the interface is more fully-featured and complex. Commercially, this course has not been a significant success. The area of "introductory Python course" is very saturated, making it difficult to become very successful.

If I were developing a Coursera course again, one thing I would do is learn how to do more of the formatting myself. I drafted the content in one place, the IETL formatted it on Coursera, and then I checked it and we iterated if necessary. Whilst my IETL colleagues were very competent and helpful, this process introduced significant extra work. If I had produced most of the content directly on Coursera, leaving only the more complex bits for my more experienced colleagues, it would have significantly reduced the overall amount of work.

Overall, I'm pleased to have developed the Coursera course. It provides a great asynchronous resource for learners to study the basics of Python. As this is one of our most popular topics, it supports a large number of students and frees up a lot of teaching time, helping to justify the large time and effort outlay of creating the course. Careful consideration of the content, sequencing, problems users might encounter, formatting and which examples and exercises would be most useful help make it a useful resource for learners.

4.4 Comparing MOOCs and online degree content

In this section, Samantha and Nick compare how content was created and is currently delivered in four MOOCs for Statistics for Public Health and the corresponding Statistics for Public Health core module (previously known as specialisation), which is part of the Master of Public Health online (MPHO) degree at Imperial College London.

Massive Online Open Courses (MOOCs) are online courses, not often associated with any formal qualifications. There is no cap on the number of students who can take these courses. Although the content could be developed by highly qualified tutors in the field, they would have little to no interaction with the students who are enrolled, and the students would need to rely more on the course content. In comparison, online degree content is content associated with a formal high-level qualification such as a certificate, diploma, or a degree. There would be similar structures to in-person degrees in place, such as an admissions process, a cap of student numbers, and two-way interactions between students and tutors. (School 2021).

Four MOOCs for Statistics in Public Health were developed in Coursera by Prof. Alex Bottle, Prof. Victoria Cornelius, and Dr. Lisa Danquah from the academic team, and Dr. Argita Zalli and Helen McKenna from what was at that time the Digital Learning Hub at Imperial College London. Each MOOC consisted of four weeks with videos and readings, some which focused on statistical concepts and others on coding in R. There were also quizzes included to reinforce concepts being learned. Each MOOC has a specific dataset, which allows students to learn statistical concepts and the corresponding R code. The MOOCs were launched in Coursera between November and December 2018. Samantha started taking care of the delivery in 2020 and Nick joined in 2023.

The MOOCs were developed for several purposes. Two important ones were to be aligned to Imperial's strategy of creating worldwide collaboration to meet global grand challenges, as the MOOCs are aimed at anyone who has an interest in learning Statistics for Public Health, and to give the learners a taste of the content in our MPHO programme. If they liked the MOOCs, they have the option to enrol and obtain a Postgraduate Certificate (1 year part-time), Diploma (2 year part-time) or the Master's degree (2 or 3 years part-time). This is well captured by several five-star reviews in Coursera. Here is an example from January 2021:

> "This is a comprehensive and well-made overview of statistical principles and techniques (1) in the context of public health and (2) that will be useful in the subsequent courses in the Coursera Specialization where it belongs. While there are a lot of similar MOOC offerings around, the public health examples and the unique approach this course provides make it worth taking especially if you are

> the type of person who wants to "cover all bases." This is highly recommended for those aiming to have a career in public health-related research or even those casual learners who want to make sense of the data that they see and hear from the news"
> — Coursera review

To access the MOOCs, there were originally two options. The first option is to obtain a certificate when paying for full access; this option is still available. This is more suitable for those students who are interested in accessing all the content asynchronously. Each MOOC was designed with four weeks of content. However, Coursera allows other timeframes to complete them. From our understanding, it is not necessary to complete MOOC 1 to move to MOOC 2 and so on.

The second option to access the MOOCs, which is the one we were suggesting to our students holding an offer for the MPHO, was to audit the whole MOOC 1 for free. In that way they could start to get familiar with the statistical concepts, follow instructions to install R/RStudio and start practising in R. However, in August 2025, Coursera replaced the auditing by a 'preview mode' which currently only allows full access to the first week of some of the MOOCs.

Table 4.1 below shows a summary of the number of students enrolled and the completion rate up to August 2025 for the four MOOCs.

Table 4.1: Summary of Students Enrolled on the Four MOOCs and the Completion Rate. Source: Coursera

MOOC No	**Title**	**Enrolled**	**Completed (%)**
1	Introduction to Statistics & Data Analysis in Public Health	60,185	10,962 (18%)
2	Linear Regression in R for Public Health	17,223	4,572 (27%)
3	Logistic Regression in R for Public Health	14,308	3,167 (22%)
4	Survival Analysis in R for Public Health	16,007	3,096 (19%)

For the degree students, there have been several additions to the content. We will provide here three pivotal ones: the different options to increase student-faculty interactions, the mock assessment, and the summative assessments.

4.4.1 Interactions between students/learners and faculty in MOOCs and degree content

There is currently no consensus on the meaning of 'engagement'. However, it has been proposed that one key engagement dimension from the National Survey of Student Engagement is student and faculty interaction (Robinson and Hullinger 2008). This dimension is clearly different between the MOOC learners and the degree students. The MOOC learners have required over the years minimal interactions with staff, and have mainly contacted us when there is an issue to be awarded a certificate. Prof. Bottle monitored the course for a brief period after the MOOCs were launched, but there has not been any reason to continue doing it. Therefore, we have minimal monitoring and modifications to the content. The higher numbers of students have also posed challenges for one-on-one engagement.

In contrast, the degree students have more frequent interactions with staff. These interactions include two induction sessions to support installation of R/RStudio (60 min each), then, during term time, 15 weekly office hours to solve statistical questions (60 min each), 12 weekly R drop-in sessions to solve coding questions (60 min each), forums to post questions related to statistical or coding questions, 13 weekly live sessions to reinforce the online learning (60 min each) and communication via e-mail. The number of degree students enrolled in the Statistics for Public Health module has varied between 2019 and 2025 from 60 to 110. As students are studying part-time and are based worldwide, it is their choice to interact with faculty members or not. Therefore, usually 10-20% have at least one student-faculty interaction. Several of these interactions have been useful to update the degree content (e.g. providing alternative commands to displaying R output in the format provided by newer versions).

4.4.2 Mock assessment

Compared to the MOOCs, MPHO students have the Statistics module with 12 weeks of compulsory content. This is the standard length of modules in the online degree to allow students time to prepare for their summative assessments. The content from the degree had modifications from the four MOOCs to cover all the essential concepts in 12 weeks. In week 13, MPHO students are provided with an optional formative mock assessment, which is not accessible to the students enrolled on the MOOCs. This is because the mock assessment, which has its own dataset, is directly linked to the summative assessment that MPHO students have to successfully complete in order to pass the module and obtain their high-level qualification. With access to the mock assessment, students can practise the skills acquired throughout the module and write an abstract with the results obtained. From the abstracts submitted each year, two of them are selected as examples and feedback is provided by staff and peers in the last online live session of the module. This allows the students to have a better understanding of how to put together their abstract for the final summative assessment. For the students whose mock assessment abstracts are chosen for the online live session, their abstracts are anonymised and the students receive detailed feedback, which is an incentive to take part.

4.4.3 Summative assessments

As mentioned earlier, in the paid version in Coursera, a certificate can be obtained from each of the four MOOCs, and to do so, different assessments have to be completed:

- For MOOC 1, seven short quizzes have to be completed (each 10%, with 3 to 6 autograded questions) and then an end-of-course quiz (30%, with 15 autograded questions).
- For MOOC 2, there are two autograded quizzes, both with 7 questions each.
- For MOOC 3, there are 5 autograded quizzes, having between 4 and 10 questions each.
- For MOOC 4, there are 4 autograded quizzes, having between 1 and 10 questions each.

The content of the assessments has remained the same since 2018. However, the content of summative assessments in the MPHO has evolved over the years. This usually takes several days to create. The first cohort had 3 autograded summative multiple-choice questions plus the main assessment, consisting of coding the best model in R to answer a research question with the dataset provided, and writing the corresponding 350-word abstract. The variety of models generated for the abstract created a challenge in terms of marking all the assessments in a standardised manner. Further, despite requiring the students to upload a data flow diagram, it was sometimes hard and time-consuming to work out what they had done, and where they had gone wrong.

Therefore, since the second cohort, the faculty team decided to generate the R code and output themselves. Students have to select, from the R output, the most appropriate model to answer the research question, write an abstract, answer 10 autograded yes/no questions on whether various parts of the output are useful, and finally answer five or six free-text questions, which mainly look at interpreting various outputs.

Recently, owing to Imperial's initiative of reducing assessment load across the University, and the rising risk of AI usage on summative assessments, we amended the summative assessment on the module. From the academic year 2025/2026, there is one summative assessment, which includes one autograded summative multiple choice questions related to the all learned content, plus interpreting and selecting the most appropriate model from the R code to answer a research question, write an abstract and answer five to six free-text questions. Currently the use of AI is not allowed in the summative assessment, hence we have followed Imperial's guidances whilst creating the assessment, to try to make it as AI-proof as possible by including some reflective free-text questions and/or questions to interpret graphs.

These changes in summative assessment have also required over the years updating a few items in the degree content, to ensure we are providing good scaffolding for students, not only to learn statistical concepts and coding in R, but also to prepare them for their summative assessments, specifically their research project where they will employ the skills learnt on the Statistics module to effectively write an abstract. The specific changes to degree content have included providing mock multiple choice questions and two readings, one with FAQs and another one describing in detail what is required in each section in the abstract, with a couple of examples of abstracts from the literature with feedback on how each can be improved according to the marking rubric.

Overall, the use of shared high quality content for both MOOC and degree learners based worldwide proved to be a successful approach. It allows flexible, accessible education to equip learners with basic statistical methods used in the public health field. Those who are interested in specific topics with minimal faculty interactions can take the MOOCs; future work will focus on increasing the amount of free content available and ensuring there are opportunities to keep the MOOCs content up to date. Those who are interested in obtaining a high-level qualification from the School of Public Health at Imperial College London have access to continuously updated content, with increased student-faculty interaction, and scaffolding for the summative assessment which at the same time empowers them to write their own abstracts.

4.5 Blended work-based learning

In this section, Joseph and Leila describe their experiences of developing and delivering online modules for the Graduate Apprenticeship and Degree Apprenticeship (GADA) BSc (Hons) programmes in the Department of Computer and Information Sciences at the University of Strathclyde.

Graduate Apprenticeship and Degree Apprenticeship (GADA) programmes are delivered over 4 years with three terms each year. They are blended work-based learning programmes, designed in partnership with industry, to allow the learners to gain a degree while in employment. GADA programmes combine largely asynchronous online learning with one day every 4 weeks of mandatory on-campus activities and one day per week of non-mandatory on-campus activities.

Since 2019, we have taught various modules in these programmes. Our blended model consists of short video lectures, screen-captures of worked examples, directed readings from textbooks, quizzes to self-evaluate learning, hands-on programming exercises, moderated online discussion forums, and on-campus activities. Both authors have contributed to the development of online materials as well as the delivery of the modules.

Before preparing online materials, staff members are offered training by the university to gain skills and experience with recording and editing. During the recording phase, each 10-minute video took approximately one hour to prepare and produce. Each week of a 20-credit module typically includes 10 to 15 videos, and the module runs over 12 weeks. Support from the university team helped with video editing, managing the Moodle page, and handling administrative tasks.

Based on our experiences of teaching these blended modules, we observed that different learning resources play complementary roles in supporting students' learning. Short videos and worked examples provide concise, targeted explanations of specific techniques or programming concepts, while textbooks offer deeper theoretical context. Practical exercises and formative quizzes enable students to test and consolidate their understanding. Online discussion forums further support learning by facilitating peer interaction, which is particularly valuable for GADA students who are balancing academic study with professional responsibilities.

The asynchronous nature of the learning materials is particularly beneficial for apprentices. Students frequently revisit video lectures and worked examples, especially in preparation for assessments. The ability to pause, reflect, and re-watch explanations allows learners to study at their own pace and accommodate the constraints of full-time employment.

Our experience also suggests that participation in online discussion forums increases when activities are clearly structured and linked to specific exercises or tasks. When prompts are broad or loosely defined, students may be unsure how to contribute, and engagement tends to remain limited. Providing clear instructions or discussion prompts helps to establish expectations and encourages more meaningful interaction.

Another important consideration is the maintenance of online content. While the creation of digital materials is initially resource-intensive, ongoing revision is equally important. In some cases, videos must be re-recorded to maintain clarity or reflect changes in software tools and programming environments. This need for regular updates is particularly relevant in computer science, where technologies evolve rapidly.

Ultimately, effective teaching depends on the ability to engage students and foster their enjoyment of learning. When learners are actively engaged, they tend to become more independent and motivated in their studies. Creating such engagement, however, remains one of the key challenges of online and blended learning environments.

Drawing on our experiences of designing and delivering these modules, we offer several practical recommendations for educators developing similar programmes.

4.5.1 Designing coherent learning pathways

Students benefit when the relationships between videos, readings, exercises, and discussion forums are made explicit. When these elements are clearly connected, learners can better understand how each activity contributes to their development. Structuring materials in smaller units is particularly helpful. Dividing longer lectures into short videos focused on individual topics enables a more balanced learning experience, allowing students to alternate between watching videos, reading supporting materials, and completing exercises.

4.5.2 Keeping video content concise and interactive

Video lectures are most effective when they remain brief and focused. Unlike traditional face-to-face lectures, online videos do not need to repeat explanations in multiple ways within a single session. Students can pause, replay segments, or consult additional resources as needed. Incorporating interactive elements within videos can further enhance engagement, particularly when introducing complex programming concepts. For example, instructors can present a short exercise during the video and encourage students to pause and attempt a solution before reviewing the worked example. Tools such as H5P[4] allow such interactions to be embedded directly into the video environment.

[4] https://h5p.org/

4.5.3 Supporting discussion and collaboration

Online discussion forums can play an important role in supporting peer learning, but they require careful design to function effectively. Broad instructions often lead to minimal engagement, as students may not know how to begin contributing. More focused prompts or discussion tasks linked to exercises or assessments tend to generate more meaningful participation. In addition, learners frequently expect instructors to remain visible in these discussions, even when the primary aim is to encourage peer-to-peer interaction.

Collaborative activities conducted outside the classroom can also enhance engagement. Off-campus group work allows students to share perspectives and experiences from their workplaces, enriching the learning process while maintaining the flexibility required by work-based learners.

4.5.4 Encouraging practice and self-assessment

Programming modules require frequent opportunities for practice. However, too many summative assessments can place considerable pressure on students who are already managing professional responsibilities. For this reason, formative exercises are particularly valuable, allowing students to experiment, make mistakes, and develop their skills without the pressure of grading.

Self-assessment activities also support learning by enabling students to evaluate their own work independently. This process helps build confidence and encourages learners to take greater responsibility for monitoring their own progress.

4.5.5 Providing timely feedback

Immediate feedback is an important motivator for learners. When students receive rapid responses to their submissions, they are more likely to correct mistakes and refine their understanding. Automated assessment tools such as Coderunner (Lobb and Harlow 2016) are particularly effective in programming modules, as they allow students to receive instant feedback on their code while reducing the marking workload for instructors. Shafti and Gemayel (2026) in this volume address the topic of automated marking in more detail.

4.5.6 Maintaining and improving learning materials

Finally, online learning materials should be reviewed and updated regularly, ideally on an annual basis. Designing content in a modular format helps make this process manageable. Short videos focusing on individual topics are easier to revise or replace without requiring the re-recording of entire lectures. Monitoring which resources students revisit most frequently can also provide valuable insights into where students encounter difficulties or require additional clarification. These insights can guide improvements to explanations, examples, and exercises in subsequent iterations of the module.

Overall, our blended-learning approach has proven highly effective for working professionals. Thoughtfully designed, developed, and maintained, it delivers both flexibility and depth, supporting sustained learning over time.

Our work with the GADA programme was acknowledged internally in 2022 when the Strathclyde GADA teaching team received the Teaching Excellence Awards Team Award, reflecting the collaborative effort and effectiveness of our blended learning approach.

4.6 Conclusion

In this chapter, we have presented several examples of online teaching. We have had different experiences, which we hope you have found useful and informative, but have also agreed on a few key points.

Online teaching excels at providing learners the ability to learn asynchronously and at their own pace. It is also the only practical way to reach large numbers of learners. However, initially creating the required materials is a big commitment, and a course requires constant monitoring and frequent updates. Videos are particularly resource-intensive and using approaches such as minimising the number of videos, or making more, shorter videos can reduce the workload in creating and maintaining content. Student engagement is encouraged through frequent exercises and tools such as automated feedback help, to make the exercises as useful and engaging as possible. For courses with more learners, automated feedback quickly becomes the only practical method of feedback.

Overall, once an online course has been created, students often find them an effective way to learn. For educators, it can provide a time-efficient way to teach, especially if the course lasts for many years.

5 Dungeon Crawlers and Anarchists: co-designing programming teaching experiences

5.1 Introduction

You find yourself in a large room with no windows; the air is still, and nothing suggests the passing of time. The door closes behind you, and it won't open for another hour or two. As you stand in front of the room, you realise there is nothing between you and what seems to be a party of the most intimidating, unpredictable creatures.

Students.

Fear not, though. You are about to learn how to deal with this kind of situation. Dealing with students' expectations (and your own) can be hard, but we believe that by working together, you will be able to deliver the learning experience that gets everyone to the next level.

This chapter is inspired in equal parts by tabletop role-playing and anarchist pedagogies. It was created to help you structuring the learning experience for your course. More specifically, we present steps you can take during an initial or preliminary learning activity for a course, preparing material that will help you:

- get to know your students, their expectations and shortcomings
- organise teams for collaborative group work
- relate learning sessions and activities to the course's learning outcomes
- structure the sequence of learning activities
- share resources

This system is inspired by variations of Role-Playing games that do not require a Dungeon Master as mediator, such as various ones from Tomkin Press[1], which reflects horizontal approaches from anarchist pedgagodies in higher education, in which students and educators operate side-by-side. (Wilkinson and Ashworth 2025).

We do acknowldge the tensions between anarchist practices and the realities of the higher education systems to which we belong, and the risk of coming across as having a somewhat

[1] https://tomkinpress.com/

inauthentic, opportunistic take on anarchism. Admitedly, we operate (and, arguably, succeed) within a system that is not as flexible, self-governing, or non-hierarchical as the one we are taking inspiration from. Still, we believe incremental, practical changes are a possible way of moving towards more inclusive and democratic realities.

> Disclaimer
>
> The collaborative, open-ended nature of some steps here described might suit workshops and short courses better than credit-bearing modules with strictly pre-defined learning outcomes. That said, syllabi are often flexible enough to allow for this ad-hoc, collaborative approach (or at least some tweaking).

5.2 Accessing the toolkit

Go to this chapter's companion repository[2]. There, you will find a toolkit for planning your teaching that includes templates for:

- Character Sheet
- Bestiary
- Adventure Map
- Party Guidelines
- Equipment List
- Skills and Spellbook

These are available as `.docx` and `.rtf` documents, but feel free to adapt them to whatever format or platform you want.

5.3 Character building: presenting learners and educators' profiles

A good first step is to understand everyone's aspirations and previous experiences. At this stage, invite all students and educators (including yourself) to complete a **Character Sheet**[3]. Once everyone has completed theirs, go through a round of introductions. Focus on understanding everyone's strong points, ambitions, and fears.

The Character Sheet could potentially help identify potential topics for class and barriers to be removed.

[2] https://zenodo.org/records/17702161

[3] The Character Sheet document was designed to help learners understand and share their own experience, needs, expectations, skills, and challenges. It is informed by Student Persona templates provided by the Open University[4] under Creative Commons Attribution-Non-Commercial licenses.

Example: Character Sheet

- *Creation: 15 min*
- *Discussion: 15 min*

Expectations:
Learning how to manage and collaborate in bigger projects using git/GitHub

Previous education and professional experiences:
Self-taught web-development.

(...)

Study Skills Weaknesses:
Might have developed bad habits from self-teaching.

The anarchist bit:

Anarchist pedagogies support and are supported by the construction of subjectivities (DeLeon 2012). Feel free to extrapolate the Character Sheet to an exploration of long-term goals, ideals, and a broader sense of identity, even if (perhaps *especially* if) they diverge from immediate, utilitarian aims of the course.
In some cases, however, discourse around subjectivities and course objectives might intersect: students might want to discuss their preferences for sectors they'd wish to work with (for instance: non-profits insted of big tech), or their affinity with open-source initiatives as a matter of citizenship and participation.

5.4 Challenges ahead: establishing learning outcomes and syllabus

Use the **Bestiary** template to establish (or remind) learners of the course learning outcomes and syllabus. Discuss what they will learn, giving examples if necessary. Depending on your course's flexibility, outcomes and topics can be decided on the spot. Usually, you will want to choose a set of learning outcomes and lessons beforehand[5].

[5] Many programmes require educators to achieve specific learning outcomes. If you are the one in charge of keeping the programme or module specification form, a good practice would be keeping it flexible enough so that new, emerging topics can be brought into the syllabus.

Example: Bestiary

- *Discussion and adjustments: 15 min*

Name:
Create GitHub repository for your project
[**x**] Main Quest [ ] Side Quest

Learning Outcomes:
1. Apply best practices to source code management and versioning

Challenge:
Follow all steps from creating a GitHub account to creating a new (empty) repository.

Relevant Equipment and Skills:
GitHub website, GitHub Desktop, Online learning material on GitHub for beginners.

The anarchist bit:

Once again, there should be plenty of space for learners to express themselves by suggesting topics for co-creating their learning experience. Such a practice is present in many other progressive pedagogies (for instance, see Freire 2017).
In practical terms, that can be very beneficial to the course. It's not uncommon that students are informed about emerging practices and tools that can enrich the course and learning experience for everyone. As educators might struggle with keeping up-to-date with professional practices, being attentive to students' perspectives can be really helpful.

5.5 The treasure map: structuring the course and learning activities

Use the **Adventure Map**[6] to discuss the sequence of learning material and activities, assessment (beasts), and how they relate to learning outcomes. Keep the discussion open so that learners can suggest, based on their expertise, perceived needs or fears, how to best organise the learning trajectory. To structure their journey, place items from the **Bestiary** into the journey map, and associating them with the course's learning outcomes.

[6] The Adventure Map document was designed to help educators and students to set out learning outcomes, syllabi, and assessments. It is informed by material created by the University of Leeds OD&PL[7].

Example: Adventure Map

- *Discussion and adjustments: 30 min*

Session	Topics	Main Quest	Sidequest(s)	Learning Outcomes
1	Source-code management	Create GitHub repository for your project	Follow another student's repository	1. Apply best practices to source code management and versioning

The anarchist bit:

The collective effort into creating a learning journey that does not follow a blueprint is characteristic of anarchist pedagogies (DeLeon 2012). Still, students and educators might benefit from some structure, in which case it is advisable to outline a learning path. Feel free, however, to continuously discuss, review, and re-negotiate that path.
That discussion could open opportunities for improving and streamlining teaching delivery. You might notice that students might need more (or less) sessions and materials on specific topics. Alternatively, you might consider making additional content available for students who need (or want) it for either keeping up with the group, or extending their knowledge.

5.6 P.A.R.T.Y: organising groups and groupwork

As a group (or groups, depending on the number of students), complete the **Party Guidelines** sheet[8], establishing ground rules on how to deal with each other and the group, as well as roles for each participant.

Go over the **Adventure Map**, discussing how to best achieve the challenges (either individually or as pairs or teams). Discuss different strategies for group work; for instance, should everyone work on every aspect of the assignment, or should they focus on their own areas of expertise and interest?

Party Guidelines should help teamwork run smoothly, including agreements for conflict resolution.

[8] The Party Guidelines document was designed to help learners manage their groupwork. They are informed by frameworks from the Institute for Academic Development[9] (University of Edinburgh 2024) and the Learn Higher CETL at the University of Bradford (Learn Higher 2012a, 2012b) under Creative Commons Attribution-Non-Commercial licences.

Example: Party Guidelines

- *Discussion and adjustments: 15 min*

Topic	Rule
What should someone do if they cannot attend a meeting or complete work on time?	Group will redistribute work according to members' strengths.

The anarchist bit:

When deciding rules for working groups, aiming for consensus would be ideal. A quick introduction and effective guide on how to achieve consensus[a] in group meetings was created by workers' co-op of campaigners Seeds for Change, and could be brought into discussion.

Conducting group work can be challenging, particularly when assessment is involved, as students might question their lack of autonomy within a group. It's worth noting, however, that: (a) individual autonomy and freedom is not necessarily the utmost important value in anarchism (see, for instance, Suissa 2006, 14); (b) although the traditional role of group leaders might harder justify, certain groups of anarchist values operated under leaders stripped down of authoritarian coercive powers (a good example being pirates as described by Graeber 2024).

In practice, students might need to be reminded on how group work allows them to develop more complex outcomes; and even though we are advocates for self-governing, there might be the need for some mediation from educators in case conflicts arise within a group.

[a] http://www.seedsforchange.org.uk/shortconsensus

5.7 A kind of magic: distributing learning material

Whether you are using a textbook, slides, PDFs, online videos, or else, use the **Skills and Spellbook** to signpost where students can find learning material that is relevant to their learning journey. Those can be tailored to individual lessons and activities, or as a general compendium (Warhammer pun not intended), and they could be presented all at once or separately as the learning advances. Again, you could populate it with resources related to students' aspirations.

Example: Skills and Spellbook

Topic:
Source code versioning

Resource Name:
Online learning material on GitHub for beginners

What it is for:
Learning the basics of GitHub for software versioning.

Study Skills Weaknesses:
Might have developed bad habits from self-teaching.

Information and Instructions:
Accessible through GitHub's YouTube playlist[a]

[a] http://www.youtube.com/playlist?list=PL0lo9MOBetEFcp4SCWinBdpml9B2U25-f

5.8 Equipment list: providing software and resources

To complete the setup, provide your group with an **Equipment List** of software and other resources they might need to use to advance the journey. You should have a preliminary list ready, but be open to suggestions from students. Below, an example that would support our student with his desire to learn source code management and versioning.

Example: Equipment List

Websites:
Name: GitHub
Purpose: Source code management and versioning
URL: https://www.github.com

Software:
Name: GitHub Desktop
Purpose: Source code versioning
URL for download or installation instructions: https://desktop.github.com/download/

5.9 Setup complete: happy adventures!

Once you have completed the previous steps, you will have generated everything you and your students need to accomplish course objectives and keep track of your achievements. At the start of each learning session, use the **Adventure Map** to update students on their next steps, and on how far they have travelled so far.

Throughout the semester, following the plan from the **Adventure Map**, present new tasks, resources, and skills, from the **Bestiary**, Equipment **List**, and **Skills and Spellbook**, respectively.

Frequently remind students to keep their **Character Sheet** updated with the new skills they acquire.

If conducting group work, remind them of the rules expressed in their **Party Guidelines** whenever there is conflict.

At the end of the course, remind students of their journey, highlighting the learning outcomes achieved along the way. Hopefully, this collaborative approach to learning will have made your job (and of your students) easier and more enjoyable. We would love to hear from you about your experiences, so please feel free to get in touch!

Happy teaching.

> The anarchist bit:
>
> Another characteristic of anarchist pedagogies, particularly in the arts (and programming is often seen as a liberal art), is how they embrace the fluidity that is necessary for making changes to the learning experience after reflection and evaluation (Wilkinson and Ashworth 2025). After running you campaign, don't be afraid of modifying, discarding, or adapting the system we just proposed according to your experience and that of your students.

5.10 Acknowledgement

The authors would like to thank Samantha Ahern and Lucia Michielin for their revision and feedback.

6 Second Languages: Teaching C to Python+Jupyter Novices

6.1 Introduction

This chapter explores challenges associated with learning a second programming language, in particular for a population of students who are not specialising in computing. In the context of our case study, the second language challenge is compounded in two ways: firstly, the programming paradigm shifts from a high-level managed interpreted language to a low-level unmanaged compiled language (i.e. from Python[1] to C[2]), and secondly, the development environment shifts from a browser-based notebook to a terminal-based text editor (i.e. from Jupyter[3] to an editor such as Vim). We highlight common misconceptions experienced by students during this transition process and discuss potential fixes, in particular the use of a *bridge language* to ease the transition from high-level to low-level paradigms.

6.1.1 About Us

Two of the authors of this chapter — Sam Skipsey and Gordon Stewart — comprise two thirds of the staff teaching the P2T course that forms much of this chapter's basis. Sam has a background in Theoretical Physics, although they have always been interested in computing as a topic; they might well have ended up doing a computer science undergraduate course had things gone differently. Gordon is a computer scientist and software engineer by training, who now works as a research IT specialist. The other two authors are based in computer science departments and are involved with teaching programming to computer science students directly. They also have interests in computational thinking.

6.1.2 Physics at The University of Glasgow

The rise of Python as the most popular programming language in the world has been coupled with the emergence of Project Jupyter and its notebooks as an extremely popular programming interface in many disciplines. It is now frequently the case (writing in 2025/26) that the only

[1] https://www.python.org
[2] https://9p.io/cm/cs/who/dmr/chist.html
[3] https://jupyter.org

experience of coding that novice programmers may have involves "Python in Jupyter", and so naturally this becomes their sole model for how programming works. The ubiquity of Jupyter also means that some students may only know "Jupyter" as a synonym for another hosted development environment, such as GitHub Codespaces[4].

As a consequence of arguments similar to those described in (Balreira et al. 2023) and elsewhere, Python has displaced other languages in many university courses, not least in physics curricula. Broadly, the suggestion is that Python's perceived relative "simplicity" in syntax allows students to focus more directly on the actual process of "programming", rather than getting hung up on syntactical complexity as in (say) C. In the University of Glasgow's School of Physics and Astronomy, students encounter programming in their first-year labs when they use short Python snippets in Jupyter notebooks to process and analyse lab results; students subsequently receive a more formal introduction to Python in Jupyter during their second-year labs. In the rest of this paper, we will use "Jupyter" as a shorthand for "Jupyter with a Python kernel"; some of the aspects here are related to Jupyter itself, but others are dependent on the particular kernel (and thus the language) used, so we wish to be clear about the conflation here.

These two lab-based experiences are, for many Physics students, their very first introduction to programming and, indeed, their first introduction to the concept that programming is important to the practice of physics as a discipline. We've noticed that this sometimes requires a significant adjustment to the students' mental image of what it means to be a physicist in the modern world, which seems to be set by prior experience in primary and secondary education as "someone who does experiments and knows advanced mathematics". Students are also expected to take mathematics courses in their first two years at university, but because they expect this as part of their image of the subject, they do not dispute the necessity of this; they may, however, dislike other aspects, for example the extreme rigour of some proof requirements in mathematical processes. By contrast, a significant fraction of physics undergraduates in their first year do find the introduction of computing topics both surprising and, if not unwelcome, then at least not well justified.

6.1.3 "P2T" – a specialised course in C for physicists

In the second semester of year two, Physics offers the course "P2T[5]: C Programming Under Linux", which introduces lower-level programming in the C language[6], the use of the GNU/Linux operating system, and command-line tooling for debugging, build orchestration and version control. This course is a prerequisite for students on the Theoretical Physics track, who have the programming background mentioned in the previous section; meanwhile, approximately half of the cohort each year is made up of second-year students from the School of Computing

[4] https://github.com/features/codespaces

[5] University of Glasgow short-codes for courses are almost opaque for historical reasons, but this was originally short for "**P**hysics level **2**: **T**heory".

[6] This is true of the 2024/25 iteration of the course, which was in place when we wrote the first draft of this chapter. We'll discuss which choice we made for 2025/26 at the end of this chapter.

Science, who begin with a much stronger background in programming concepts, including experience of assembly language. In fact, P2T is a small (10 credit) course designed to stand alone with no formal prerequisites and thus is available to any student; in practice, however, only a few students outside of physics and computing science choose to take it.

When first developed, at the turn of the millennium, P2T was considered a rather specialised course and was delivered to a small cohort. Today, mostly due to interest from computing students, it has grown to around seventy students a year; for comparison, the total number of students studying second-year physics and computing science is approximately 180 and 330 respectively.

We have observed that the difficulties experienced by the physics cohort in P2T have changed as their prior experience has become dominated by Jupyter. In this chapter, we will discuss how this relates to the mental models formed when learning to program in Jupyter, and their differences from the models needed for lower-level programming. We will also suggest some potential bridge languages that would ease the transition and help students to develop better mental models.

6.2 Misconceptions + Difficulties

This section comprises a brief list of the most common conceptual challenges we observe in students used to Jupyter when learning C in P2T. Quoted text is paraphrase from a genuine student question, other titled sections are summaries of the general area of misapprehension. There is no implication in the ordering of the examples.

1. “If you have a test (e.g. $3 < 1$) which is false, does that make the compiler stop or the program stop?”

   This is quite a subtle distinction, which is less evident perhaps when learning first in an entirely interpreted interactive programming environment such as Jupyter, where code-blocks themselves obscure the flow of execution, let alone the distinction between compilation / interpretation and execution itself.

2. Source Code == Executable Code

   More generally, we often observe confusion between a program’s code, its executable, and a process which represents a running instance of the program. The very first C exercise in the P2T labs walks students through the process of fixing some provided code, introducing an iterative sequence of compile-fix-repeat, and finally executing the compiled program. Despite this, towards the middle of the course we usually find that a significant number of students start trying to execute their source code, often after compiling it. In the case of P2T, this confusion is exacerbated by the fact that students are introduced to shell scripting in Bash, an interpreted language, around the time they develop this confusion.

3. Import == #include

   In order to do most useful things in C, we start students off with the magic incantation `#include <stdio.h>` and explain that this enables them to use functions that interact with the terminal, such as printing out text or reading input. We do allude to the fact that `#include` only provides half of the functionality that `import` does in Python, but when we introduce the full concepts of header files and libraries later in the course, students find this hard to reconcile with their idea of `import`. In particular, the need for a linking step is an alien concept.

4. Types

   Types represent a significant learning curve for our students, especially those who are entirely new to programming, or who been introduced to Python via the physics educational flow (despite being encouraged to use NumPy almost immediately in that case). Students' misapprehensions in this category are varied, ranging from the difference between variable declaration and assignment (in C and more strongly-typed languages, declaration requires the type of a variable to be specified, while assignment does not) through allowable type conversions and the specific differences between different kinds of numeric scalar types, to uncertainty about the role and use of derived types.

5. CStrings and CArrays == Lists... and C types "are objects"

   In a sense, this is a subcategory of misconception 4, but it feels distinct because Python so strongly centres dynamic lists as the default container with a syntax strongly resembling that of C's decidedly static and strongly-typed arrays. This results in a number of conceptual problems for students either directly — trying to append to, slice or copy-by-assignment arrays — or indirectly trying to apply Python's "for-each" loop semantics to C via implicit array iteration (that is, writing a loop as a "for each i in CArray").

6. Scope and Lifetime / Encapsulation

   Python does have rules for both the scope of name bindings and the lifetime of allocated variables, albeit via garbage collection in the latter case, but the pragmatic way in which Python is taught in first and second year — and moreover the confusing way in which Jupyter cells interact with these concepts — means that students have not been exposed to these prior to starting P2T.

6.3 Mental Models

None of the above misconceptions are the students' fault: most are perfectly reasonable generalisations from an initial programming experience involving only Python in Jupyter, which naturally insulates developers from some of the underlying mechanics required to translate a series of high-level source code excerpts into a program that a computer can actually execute (Johnson et al. 2022).

6.3.1 Blaming the Snake or the Planet

It's important to spend some time teasing apart the relative contributions of the notebook context and the language itself in forming these misconceptions. As one of the authors has noted previously (Singer 2020), there are specific pedagogic *disadvantages* to the notebook approach which have been understood for some time, and long pre-date the advent of Jupyter; one of the authors remembers some of these issues arising when using *Maple* in the late 1990s!

Broadly, we observe that notebooks in particular make it very hard for students to develop mental models of the flow of execution through code, and the way in which state is transformed in this process. There are notebooks — for example, the *Pluto* notebook implemented in Julia — which partially fix this problem. Pluto addresses this issue by specifically re-executing *all* cells by default, in order, when one cell is modified. However, Jupyter remains the mainstream choice of notebook platform, by sheer weight of deployments. Thus, it may be obvious that it is the use of Jupyter, not Python, that is the primary cause of misconception 1, and that Jupyter contributes to some significant extent to misconception 6.

Misconception 2 may also be caused by the use of Jupyter to a greater extent than the use of Python. Students also find it hard to reason about files as objects, a well-documented issue as increasingly locked-down operating systems prefer to present search tools and containerised applications with their own local files, rather than the OS's own filesystem (Chin, n.d.)[7]. The use of Jupyter correlates with this, as notebooks are both seen as more natural for students used to browser interfaces (as has become increasingly common over the 2010s and 2020s) *and* themselves promote that same disconnection of content from location. In this sense, we believe that misconception 2 is promoted less by the fact that interpreted languages are not translated into a permanent, separate, executable artifact, and more because even the concept of code as living in static, separate files is elided by the interface. Of course, this issue might also be raised with any REPL, but the image of the modern web notebook as "just another browser app" surely does not help matters.

Misconceptions 3, 4 and 5, conversely, clearly originate with the language itself, or at least with the pedagogic approach adopted by courses the students have previously undertaken.

6.3.2 A "Whorfian" third way

Pedagogically, the programming languages we teach students in a specific discipline should be chosen to promote particular modes of thought. Pragmatically, we are also pressured to teach languages which provide students with employable skills, and these two goals may not perfectly overlap. Esoteric programming languages, for example, can help to develop specific problem solving or mental models in programmers (Singer and Draper 2025), but are by definition not widely used.

[7] Tangentially, we also see students confused by the "content" of a file not being a function of its filename suffix: an internalised Windowsism.

For our physics and computing science cohorts, there are therefore different reasons to wish to teach C in the first place:

Physics students:

- To develop mental models of low-level interfaces, important for those students who will develop experimental skills with electronic data taking, or hardware design; for example, the design of particle detectors may involve firmware development and FPGA programming.
- To give experience in performance programming in a language in which many core scientific libraries are still written.

Computing students:

- To develop mental models extending their knowledge of machine behaviour from assembly courses studied in first year, and feeding into low-level topics covered at honours level, such as operating system and compiler design.
- To appreciate that the Python runtime abstracts away many aspects of program behaviour, including dynamic memory management.

As such, appropriate mechanisms for developing those mental models may differ between the two cohorts. We explore two potential 'bridge' languages to scaffold the mental models for students before their introduction to C: one already existing for computing science students, and one proposed for physicists that may even be able to supplant C for many of these students.

6.4 Bridge Language: Java

In the School of Computing Science at the University of Glasgow, students learn Python in their first year of undergraduate study, alongside a specialised assembly language designed specifically for teaching. In their second year, students undertake several courses using the Java language, most notably a course on Object-Oriented Software Engineering. By the time they begin P2T in the second half of this year, they have been exposed to multiple languages for several months, and have started to develop the agility needed to adapt their mental models appropriately for whichever language they happen to be using at the time (Tshukudu et al. 2021).

The choice of Java may be particularly useful as a bridge between the conceptual frameworks implied by Python and those required for efficient development of code in C and C-like languages. In some ways, we can consider Java to be a halfway house: like Python, it is deeply object-oriented (in the C++ sense), to the extent that there are classes that wrap even the primitive fundamental types; like C, it is strongly-typed and its default containers are statically-sized, although Java does provide dynamic, heap-allocated containers in its standard library. Furthermore, Java syntax superficially resembles that of C and C++, further easing the later transition to C.

6.5 Alternative Bridge Language: Julia

We suggest that, for physicists in particular, there are other languages which are equally or indeed more compelling as bridge languages for low-level concepts.

Julia (Bezanson et al. 2017) is a just-ahead-of-time (JAOT) compiled, strongly-typed language significantly influenced by R, MATLAB and other languages oriented towards engineering and statistics. This influence means that much of the notation is designed to be familiar to mathematicians, and functionality for array programming concepts and statistical data processing is built in to the language, rather than being a feature of a supplementary package like NumPy. This has led to Julia becoming the teaching language of choice for a number of quantitative disciplines — for example, the University of Michigan features Julia in its theoretical courses in robotics (Grizzle, n.d., and others) — and has also seen support for it grow in other disciplines, including high-energy physics (Stewart, Graeme Andrew et al. 2025).

Indeed, one of the issues with the arguments for Python and against performance languages, such as those given in (Balreira et al. 2023), is that they beg the question, assuming that by definition a "performance" language must also be inherently less syntactically and semantically natural than an "intuitive" one (such as, it is argued, Python). This is neither obvious, nor actually true in the general case; C and C++ are arguably very unapproachable languages, but this is a consequence of their age and accretion of features more than it is due to their performance characteristics.

The power of Julia as a bridge language from Python is that one can gradually add more layers of subtlety and complexity as students develop their mental models. The equivalent process in Python requires learning large external modules such as NumPy, which effectively provide an additional domain-specific language wrapped in Python for dealing with these concepts. In Julia, such concepts are supported within the core of the language itself.

A common pattern in development in Julia is to write functions in a general sense first, with no type specifiers on their parameters:

```julia
"quotient(numerator, denominator) : Quotient of two values"
function quotient(numerator, denominator)
    numerator / denominator
end
```

Then, when necessary, one can specialise methods of the function for particular cases:

```julia
"quotient(numerator::Integer, denominator::Integer) : Quotient of two Integers - returns a Ra
function quotient(numerator::Integer, denominator::Integer)
    numerator//denominator
end
```

(Here we specialise quotient for any concrete Integer type variables to return a Rational fraction, rather than a simple division.)

This provides a natural way to introduce types and their relevance to students. In fact, internally Julia always specialises a function invocation by the types of the argument, and we can also introspect this if we want to demonstrate this fact to students.

Of course, Python does provide *type hints* as an optional feature from version 3.9. Unlike true type constraints, however, Python's hints have no effect on the Python interpreter itself; they are annotations to be parsed by external linters, or to act as additional documentation for the human reader. The actual output of the Python interpreter itself is unchanged by their introduction.

Furthermore, as almost all of Julia's standard library is itself implemented in Julia, Julia's introspection tools can provide a way for learners to look under the bonnet and explore the internals of the language. Unlike Python, there are almost no black-boxes that we can't examine. For example: the @less macro allows the student to look at the source code for any function that is loaded from a file in the current session. While this doesn't work for things defined in the REPL itself, there are packages that provide this functionality should it be needed.

```
@less isodd(3)

> isodd(n::Real) = isinteger(n) && !iszero(rem(Integer(n), 2))
```

(In fact, a view of the file in which this is defined, with the above line highlighted.)

Teaching performance programming in Python is — by contrast with teaching fundamental programming concepts — quixotically harder, in that much of the answer is to use Python fundamental constructs as rarely as possible. Whilst algorithmic design plays some part, the fact that we can make an implementation orders of magnitude faster by using NumPy intrinsics instead of for-loops muddies the lesson that we need students to extract from the experience. By contrast, in both C and Julia, implementations in the base language are about as fast as they can be. Efficiency is almost always gained by either algorithmic improvement or, in extremis, by making use of deep systems knowledge (cache-width, layout of data in memory, etc.) which is mostly language-agnostic, and can be relegated to a later course.

Because Julia *is* JAOT-compiled, however, lower-level representations are always available to the programmer, which can be useful for pedagogic purposes; something which is not possible with Python. That is, we can always use the @code_ family of macros to expose what Julia code is compiled to, in an active session, at various levels in the compilation process, including at that of the native assembly code (via @code_native). For example, this lets us show the significant changes in code generation when switching between floating-point and integer types for mathematical operations, or how tail-call recursion is optimised to more efficient loops!

As a result, Julia is both a more complete bridge-language towards low-level programming concepts than Java (or directly C) *and* a more suitable *single* language for teaching students in physics and other mathematics-oriented disciplines than Python + NumPy (+ Numba + …).

Of course, Julia is not a perfect language. In terms of popularity, it is still rather specialist; it is considerably less popular than Python, and less popular than Python + NumPy in physics circles. This is partly attributable to its relative youth — only 14 years to Python's 35 — but also due to network effects in language adoption. That youth, and a small development team relative to Python, also means that Julia's development cycles can still introduce more significant changes than the relatively stable Python 3. (Python, of course, passed through this phase during its 2 to 3 transition, when it was a similar age to Julia now.) This presents some difficulty in maintaining static teaching materials for some topics; automatic differentiation tools, for example, have been in some flux for the past year or so, due to changes in the language internals.

6.6 Alternative Bridge: Just Drop Jupyter

As roughly half of the misconceptions we note in this article are at least somewhat due to the abstraction and disconnect that Jupyter itself brings to the coding experience, a third and perhaps less dramatic choice might be simply to teach Python as a native language in a terminal context.

The Python interpreter is ubiquitous, especially now that Microsoft's significant interest in the language has led to it being available to script Excel workbooks; in MacOS and Linux contexts, of course, it is almost always available as a system component as well.

Writing Python code in actual files enforces structural discipline that isn't present when using notebooks, and also provides at least some reinforcement of the difference between source code and the thing that's actually doing the work. Indeed, one can wave at `.pyc` files if we wish to show a transformed output.

Other misconceptions in our list are not ameliorated by this approach, of course; in particular, misconception 4 is only partially addressable, and 5 is arguably intractable as C and Python simply have different object models. However, in a physics context, Python is almost always used with an implicitly-loaded NumPy package to handle scientific data. NumPy, in contrast to Python, cares deeply about types, and also enables new iterative constructs not found in the base language, which are broadly those constructs typical of array programming languages.

6.7 Conclusions

An important step towards students reaching competency in a programming language is to develop a mental model of the way in which the code one writes becomes a series of instructions that a computer can execute. While environments such as Jupyter undoubtedly provide an accessible interface to programming tools, this accessibility can come at the cost of rendering the internal workings of the languages opaque, which can make it harder for students to gain the insight necessary to progress to more advanced domains. Rather than attempt to move straight from interactive Python development in Jupyter to a low-level language such as C, it may be fruitful first to introduce a bridge language that allows students to explore more advanced aspects of programming within a modern environment.

The choice of bridge language can be discipline-specific, however, since different disciplines program in different contexts. Whilst there are arguments for languages such as Java for computing science students, we suggest that choosing superficially more "esoteric" languages, such as Julia, may be a better pedagogic fit for physics students. If that suggestion is considered too radical, the alternative of simply teaching Python *without* Jupyter may help to ground student understanding in ways useful to their future development. This is particularly true in a scientific context where "Python" is actually taken to mean "Python with some combination of NumPy / SciPy / Pandas / <insert your scientific library of choice here>".

6.8 Postscriptum

After this chapter was drafted, the decision was made to change the contents of P2T. The current version, which is in progress at the time of writing this postscript, broadly follows the "Just Drop Jupyter" approach. Whilst the pedagogic advantages of Julia were recognised by the relevant committees, it was felt that retaining a through-line with Python was more generally useful, and also allowed a wider subsection of academic staff to be able to teach the course.

In exchange, the course now covers software engineering concepts more substantially, with lab material on unit testing, performance profiling and reproducible code and packaging now provided. The performance profiling aspects have been the most challenging, in overcoming the intrinsic "just translate it to NumPy" problem, but we have taken the opportunity to introduce the physicists to some light conceptual exercises on the scaling of different data structures as part of this.

Interestingly, immediate feedback from the physics students who had taken the previous version of course has been mixed: those who felt the need to comment mostly argued that learning a performance language like C had been valuable to their deeper understanding of programming as a discipline. It is, of course, too early to say if this is a wholly successful change, although interim feedback has been generally positive… except, again, for a small number who were hoping to learn C!

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

7.1 Introduction

We live in a data driven world. Much research in STEM departments requires skills in data description, prediction and inference which are often developed through statistics modules but, increasingly, non-trivial statistical analysis requires computer programming. Therefore, for many undergraduate degree programmes, two closely interrelated but technically distinct disciplines - programming and statistics - must often both be taught. There are different possible approaches to teaching these subjects, most notably whether to deliver them simultaneously or sequentially. In this chapter, we focus on undergraduate level teaching in subject areas that are not primarily programming or statistics. We discuss the advantages and disadvantages to three different approaches and provide guidance on best practice in instructional technique and curriculum design. Case-study examples based on our own teaching experiences are provided throughout to illustrate these strategies.

The topic of teaching statistics to non-specialists has been discussed at length (Kelly 1992; Mustafa 1996; Metz 2008; Gimenez et al. 2013; O'Hara 2016; Bromage et al. 2022). How to do so while acknowledging that programming is an important and related skill has received less focus, which we address here. In all scenarios teachers should be mindful of their own educational context, and, inter alia, what knowledge and skills they want their students to obtain from learning both programming and statistics.

7.2 Teaching statistics before programming

This is perhaps the original or "old school" approach. As authors, many of us were taught in this way in our own undergraduate degrees prior to the widespread use of high quality, open-source programming languages (e.g., Python, R). Drawing on our experience, we have found that a key advantage of this approach is its focus on core theoretical concepts in statistics (e.g., probability distributions, uncertainty) without the additional cognitive load associated with technology or implementation. In disciplines where students are likely to need to perform statistical analyses across different contexts and using a range of technologies, this emphasis on foundational statistical reasoning has been particularly valuable.

In practice, we have found that this approach cultivates in students both the ability to conduct 'back-of-the-envelope' analyses in settings where computers are not always available (such as during fieldwork), and enables them to 'think on their feet' when off-the-shelf methods aren't suitable. Importantly, we have observed these behaviours emerge spontaneously rather than through explicit scaffolding.

A second benefit of this approach is that it foregrounds statistics as a discipline in its own right, independent of computation, which can be obscured when programming and statistics are taught together. However, in undergraduate contexts where the primary subject area is not statistics or programming, separating the two can lead to the links between statistics and programming being weakened, potentially reducing insight into the applications of both. This has reinforced for us the benefit of applying statistical understanding once a conceptual foundation has been established. Indeed, if we choose a statistics-first approach then we strive to ensure that students are afforded opportunities to use computational approaches (e.g., programming or domain-specific interfaces such as STATA) to put their statistical understanding into practice at a later date.

> Example
>
> In a first-year geoscience course, statistics was taught independently of programming before moving to a combined approach using Python. Later in the year, during fieldwork at the "Ammonite Pavement" in Dorset, students were asked to measure the size of some ammonite fossils. As they were comfortable with pen-and-paper statistical analysis, students took the initiative to more rigorously characterise the distribution of ammonite sizes, and to estimate the number of observations required to obtain a representative sample. Whilst the statistics course subsequently saw students applying programming to address statistical questions, this in-the-field investigation was facilitated by students first learning to do statistics by hand. Student engagement with 'back-of-the-envelope' statistics in the field emerged autonomously, but as a result we would consider highlighting the potential benefit of this sequencing explicitly for all students.

7.3 Teaching programming before statistics

An alternative approach we have adopted in some contexts is to teach programming first, followed by statistics lessons that make explicit use of students' programming skills. Here, students have time to consolidate their programming skills and build their programming mindset. They can then build knowledge and understanding through computational investigation of statistical concepts, before being introduced to formal statistical definitions.

This approach requires more curriculum time than many programmes can accommodate. It also requires that the programming and statistics courses are designed with interaction in mind. For example, using the same programming language in both courses, or intentionally keeping the programming course language-agnostic (e.g., giving examples in multiple languages). We suggest that if this alignment is absent, students may find it more challenging to apply their programming knowledge to statistical contexts.

We have also observed that teaching statistics in a manner which is reliant on the use of programming can present a barrier for students who are less confident in this, particularly early in the curriculum. Indeed, depending on the overall curriculum design, students may have forgotten much of the programming learnt by the time they are learning statistics. We have particularly observed this skill fade at the start of a subsequent academic year following a long summer break. This has led us to conclude that this approach is most effective when programming instruction is closely timed with subsequent statistical teaching, and that particularly between years there is the need to build in time for refresher training. We would avoid a programming-first design when coordination across modules is weak or when a long gap makes knowledge decay likely.

Example

In a second-year course in biomedical informatics, the concept of variance was introduced using a programming-first approach prior to presenting the formal statistical framework of Analysis of Variance (ANOVA). Students were first shown how to use simulations (sampling with replacement) to compare differences between pairs of samples within and across treatment groups. These empirical results were then used to derive the probability of observing a difference between groups and to introduce the concept of variance as a quantitative measure. In the following week, the formal mathematics underlying ANOVA were introduced. Students subsequently learnt how to implement ANOVA using R and, when appropriate, to perform post-hoc pairwise tests to interpret observed differences among treatment groups. We found that this deliberate sequencing helped students to comprehend the underlying theory before learning how to implement the analysis and interpret the output in real-world scenarios. Of course, a trade-off in this approach was the additional contact time needed to run simulations and then later revisit notation, requiring a reduction of coverage elsewhere.

7.4 Teaching programming and statistics together

The final approach we have identified is to teach programming and statistics together (Sarvary 2014). In practice, we have seen this implemented in two ways: within a single module, instructors can teach either programming or statistics first and immediately apply it to the other, or the two can be truly interleaved e.g., by introducing new programming constructs and statistical concepts in parallel.

This integrated approach has clear pedagogical advantages. Most importantly, students have reported that they feel able to engage immediately with statistical concepts through computation, and that this can promote a deeper understanding of statistics and its application. Furthermore, this can be supported by introducing new programming approaches when required alongside novel statistical techniques. Students may take a constructivist learning approach to building their statistical knowledge, e.g., by running simulations and adapting code provided in class. For students whose prior negative experience with mathematics presents a barrier to learning statistics (Pletzer et al. 2010), we have found that introducing statistical concepts through code snippets can reduce this anxiety compared to presenting the same idea using formal mathematical notation. That said, our experience also highlights that for some the opposite is true i.e., that they find it easier to explore concepts programmatically after a formal mathematical introduction.

Despite the benefits of this approach, we have found that the principal challenge of teaching programming and statistics together is the risk of cognitive overload. When both topics are taught concurrently, students are not always able to consolidate new material into their long-term memory and thus the working-memory load is higher (Sweller 2018). In practice, many students will experience hurdles in their learning of both statistics and programming, and these may compound each other to further impede student attainment. We have observed that this effect can be exacerbated for students who are not learning in their native language, as they need to acquire domain-specific vocabulary in two topics simultaneously.

While thoughtful course design - such as carefully and explicitly scaffolding topics, and limiting weekly learning outcomes - can mitigate cognitive overload, it is likely to remain higher when teaching programming and statistics together than with standalone courses (Auker and Barthelmess 2020b). Additional trade-offs have also emerged. In some implementations, students' development of broader computational or algorithmic thinking was reduced, since programming was used almost exclusively to perform statistics. For disciplines where programming is primarily used for this purpose, this may be an appropriate outcome. However, where courses aim to set students up to progress to higher-level programming courses, we have found that students benefit from learning to apply programming skills across a broader range of applications to widen their transferable competence. Finally, the choice of programming language and environment can substantially impact learning outcomes, both positively and negatively. This highlights the need for careful course design, e.g., choosing a beginner-friendly programming language when adopting a fully integrated approach. We would therefore recommend avoiding a fully integrated design when class time is too tight, or when support for learners studying in a non-native language is limited.

Example

In a first-year undergraduate course with students from several mathematically oriented degree programmes, concepts such as linear modeling were introduced alongside their computational implementation. During lectures, slides either showed the mathematical and code representations side-by-side or sequentially within the same session, allowing students to make connections across the two representations.

Students then participated in workshops where context-specific exercises with new datasets were given, requiring students to start afresh each time. Students needed to either adapt code snippets based on the given data, or write new code to address each task. We found that regularly changing datasets encouraged transfer of knowledge rather than rote replication. We also learnt that with this style of delivery it was necessary to provide sufficient pauses - either during or between classes - to allow students to fully digest the content from both disciplines, meaning that less time was available for other activities.

7.5 Practical strategies for teaching statistics and programming

Drawing on our teaching experience across a range of programmes, we now summarise a number of practical strategies that we have found effective when teaching programming and statistics, regardless of whether these topics are taught sequentially or simultaneously.

First, we have learnt that embedding the teaching of these topics into the subject matter of the overarching course is critical for student engagement, particularly when programming or statistics are not the principal focus of the degree programme. This embedding may take multiple forms, such as creating appealing visualisations, using authentic datasets, or discussing the conclusions that can be obtained in the students' particular research field. In our experience, this contextualisation helps students understand not only how to perform relevant analyses, but why they are important for their discipline.

Second, statistical methodologies are most effectively taught when also described clearly using a computational approach. For example, while it is helpful to describe the formal mathematical derivation for generating a test statistic, we recommend that this is routinely followed up with the programming approach required to generate the same result. Demonstrating the same concept from both perspectives helps students to consolidate the underlying statistical concepts and to see how statistics are applied in practice, highlighting the close connection between the two approaches.

A third strategy that has emerged from our teaching is to foster a "programming or coding mindset". We found it important to foreground programming as an important transferable skill in its own right, rather than solely being a tool to perform statistical analysis. This involved teaching students how to interact with data programmatically rather than relying on graphical user interfaces or apps alone. In practice, this included providing explicit instruction in code annotation, file paths, naming conventions, and being able to discriminate between characters, numbers, and special symbols. We also found that dedicating time to debugging code — particularly through live programming in lectures — helps to demystify programming and normalise common programming errors.

Finally, our experience strongly suggests that motivating and enthusing students about both statistics and programming is critical. For example, framing computational statistics as a means to increase reproducibility, transparency, and confidence in scientific discoveries helps students appreciate its broader value. We have also found that highlighting the fact that these skills are universally applicable, lucrative, and in high demand in the job market, including beyond academia, is useful for motivating engagement. As an illustrative example, note that the gap in data-driven technical skills reported in 2023 in the UK includes specialist data skills (including statistics), alongside programming and software engineering skills required to manage and analyse datasets (Fearns et al. 2023).

Example

In a postgraduate course covering both Python and R, we introduced a recurring activity designed to foster a "coding mindset", called 'Error of the Week'. This feature celebrates errors as learning opportunities rather than sources of embarrassment or frustration. Students were encouraged to share errors on a discussion board, including solutions if applicable. As a course team we responded to every post, and during live sessions the course lead or tutor highlighted a particularly interesting or common error to the class, debugging it live. Over the duration of the course, these discussion boards evolved into a common error glossary for all. We found that these strategies lowered the psychological barrier that programming can often present to students, allowing for more time to be devoted to understanding and interpreting statistical outputs.

7.6 Conclusion

We have outlined three approaches for teaching programming and statistics: teaching statistics first; teaching programming first; and teaching both skills together. Our experiences suggest that each approach has clear strengths and limitations, and will be best suited in different settings. In deciding which approach is most suitable it is important to consider the desired learning outcomes, overall programme design, and the cognitive load placed on students, all of which will vary depending on educational context and prior student experience.

Teaching statistics before programming is particularly useful when the priority is to develop strong, transferable statistical reasoning. Teaching programming before statistics can work well when early development of computational confidence is a key goal, and when there is sufficient time and alignment to support knowledge transfer. With careful instructional design, and sufficient curriculum time, many of the affective barriers students experience when learning programming and statistics can be mitigated. Under these conditions, our preferred approach would be to teach programming and statistics together, as this enables students to engage immediately with statistical concepts through computation, and develop a complementary understanding of both. We hope that by highlighting the advantages and disadvantages of different approaches and suggesting practical teaching strategies, this chapter will help educators to make informed decisions about how best to teach and encourage the learning of programming and statistics to their students.

7.7 TODO 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

8 Where do I even start with Pair Programming in my classroom? Conversation with seasoned practitioners.

8.1 Context

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

You can also listen to the audio podcast version of this interview[1]. Thank you to Miłosz Karpowicz and Jan Banaś for audio editing advice and help.

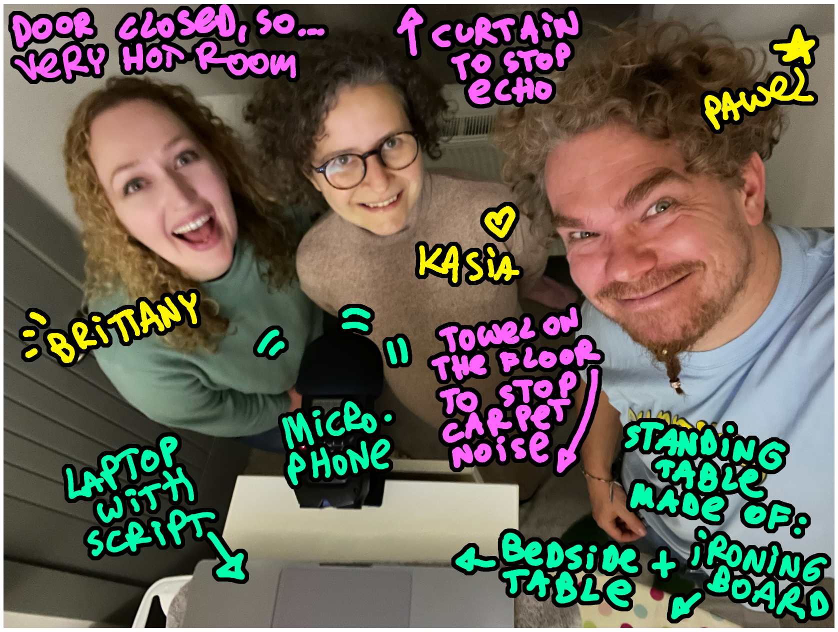
DOOR CLOSED, SO... VERY HOT ROOM
CURTAIN TO STOP ECHO
PAWEL
KASIA
BRITTANY
TOWEL ON THE FLOOR TO STOP CARPET NOISE
MICRO-PHONE
LAPTOP WITH SCRIPT
STANDING TABLE MADE OF:
BEDSIDE TABLE + IRONING BOARD

Figure 8.1: Authors recording the podcast version. We rigged the space for optimal audio quality and comfort.

[1] https://media.ed.ac.uk/media/1_xnjvow1u

8.2 Communicating the why

8.2.1 Let's jump right in:

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

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

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

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

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

8.2.2 What is Pair Programming?

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

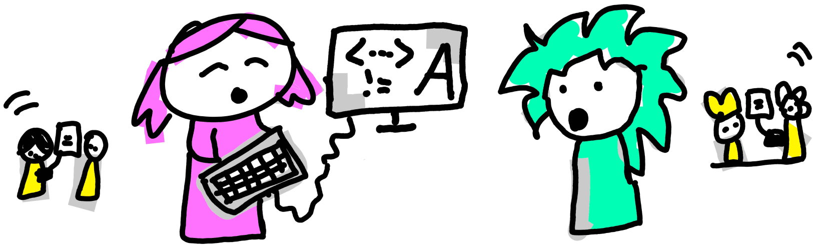

Figure 8.2: In pair programming two students work on one computer: Driver operates the computer, while Navigator helps them to find problems and offers advice.

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

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

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

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

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

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

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

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

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

8.2.3 Why PP is good for learning

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

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

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

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

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

Pawel: Totally, it gets even better - you come to practice not just with the teacher but also with all your fellow students. Everyone becomes a little bit of a teacher, and a little bit of a student. It's like we do not have a classroom of 40 students and 1 teacher... It's more of a classroom with 41 teachers - everybody contributes! And that's where doing practical things works well - no one in the class has a full grasp of the concept yet, but we will all figure it out together.

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

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

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

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

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

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

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

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

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

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

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

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

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

8.2.4 Students' reactions

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

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

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

Interviewer: Have you ever encountered students who didn't want to be put in pairs, and insisted on working alone? What do you do then?

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

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

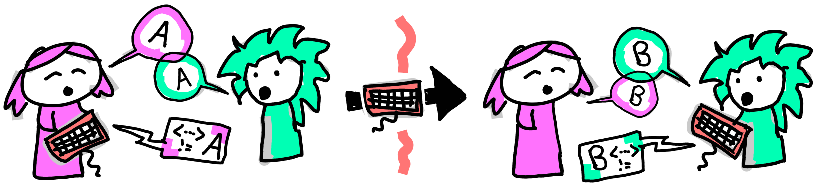

Figure 8.3: Students swap roles frequently to have experience of driving and navigating

8.3 Preparing for the session

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

8.3.1 How to run a PP session

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

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

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

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

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

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

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

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

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

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

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

Interviewer: Ok I understand the logistics a bit better now. And you mentioned that some tasks are better than others for pair programming?

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

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

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

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

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

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

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

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

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

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

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

8.3.2 Logistics (room, equipment)

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

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

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

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

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

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

Brittany: Over the last decade we've tried every possible combination. During the pandemic we had to suddenly start teaching online, including with pair programming, and it turned out to be very manageable, and in some ways even easier to run. For example, each group had its own breakout room, and the teacher could visit them with a click of a button. The students could also use the raise-hand feature to indicate they needed help. Instead of sharing a laptop, the driver would share their screen to show what they were doing.

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

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

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

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

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

8.3.3 Tasks students work on

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

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

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

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

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

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

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

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

8.3.4 Creating Pairs

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

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

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

Pawel: And in terms of how exactly we come up with pairs… we use different strategies for that. Often it is just purely random, we shuffle a list of names and that decides who's with who. And sometimes it's quite fancy, and we can be very thoughtful about creating the most diverse pairs. Since we're all geeks, we even wrote a website which creates perfect pairs for you![2].

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

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

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

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

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

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

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

8.4 Running a session

8.4.1 Before we start / Preparing the group

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

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

first few times?

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

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

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

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

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

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

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

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

8.4.2 Once pairs are at work / Roles of instructors and students

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

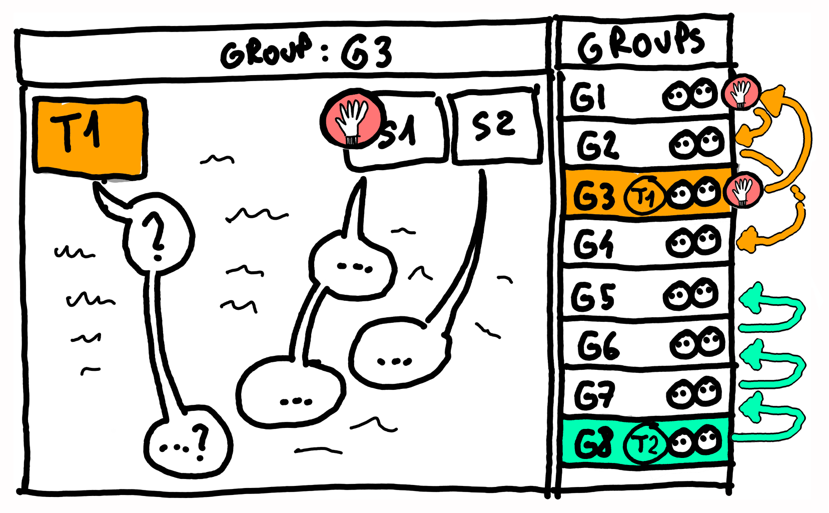

Figure 8.4: In online pair programming instructors continuously roam between breakout rooms, prioritising those with raised hands. Instructions take 'ownership' of parts of 'the room'. When instructor enters a breakout room, they first ask students to describe the problem and what was already tried. After that instructor will gives help students.

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

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

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

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

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

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

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

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

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

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

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

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

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

Pawel: And coming back to the logistics of asking for help, students who need it would just raise their hand. Some classes use the Software Carpentries[3] model where students can stick a red or green post-it note to their screen, so teachers know how they are doing, and if they need help.

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

8.4.3 End of the session

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

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

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

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

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

[3] https://carpentries.org/

All you'll need to start is: a willing colleague or friend; two hours of time; and a simple programming task (maybe a challenge from Tidy Tuesday, or Advent of Code). You could even come to one of our taster sessions which we run at conferences and such, where we pair you up with another participant, we give you something to do and let you Pair Program! You can find instructions on how to run such a workshop by yourself on our website[4].

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

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

Interviewer: This sounds like very useful advice and first steps! And if people want to know more, they can find out about it on the website of our community, `pairprogramming.ed.ac.uk`[5]! Thanks for talking to me today. I am Dr Kasia Banas, and I've been talking today with Dr Brittany Blankinship and Dr Pawel Orzechowski from the Edinburgh Medical School at the University of Edinburgh.

Brittany: Thanks for having us!

Pawel: Thank you!

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

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

9 Removing Barriers by Programming Without Computers

9.1 Introduction

Computational thinking (CT) has increasingly been recognised as a key competence for learners in the 21st century (Wing 2006). It is not limited to computer scientists but has been identified as a "universally applicable attitude and skill set" (Wing 2006). CT involves problem-solving processes that enable individuals to approach complex challenges by decomposing them into smaller parts, designing algorithms, recognising patterns, abstracting irrelevant details, and applying logical reasoning (Aho 2012). These skills are foundational for programming but are also valuable in wider contexts such as mathematics, engineering, social sciences, and everyday problem-solving (Tedre and Denning 2016).

Despite its importance, the teaching and learning of programming and CT is challenging. For many learners, the initial encounter with programming involves exposure to a programming language and development environment. While these tools are essential for producing executable code, they can impose significant barriers to entry. Syntax errors, cryptic error messages, and the abstract nature of code execution often discourage beginners (Robins et al. 2003) and many students focus narrowly on the mechanics of a programming language, rather than grasping the underlying concepts of problem-solving and algorithmic thinking (Koulouri et al. 2014).

These difficulties are not confined to school-aged learners. Lifelong learners, professionals seeking to re-skill, and students from non-computing disciplines also experience barriers when first attempting to program. Studies have highlighted that learners without prior computing exposure often report low confidence, high anxiety, and a perception that programming is inaccessible or only suitable for "technical people" (Lye and Koh 2014). Such perceptions contribute to underrepresentation of many groups in computing, particularly women, mature learners, and students from non-traditional backgrounds (Sentance and Csizmadia 2017).

Another barrier is the mismatch between learners' expectations and the nature of programming tasks. In formal education, assessments often emphasise correctness of code rather than exploration of ideas. Learners may therefore see programming as a high-stakes activity where failure is punished rather than as an opportunity for experimentation and iterative problem-solving (Watson and Li 2014). This can lead to disengagement, particularly when feedback comes in the form of obscure compiler errors rather than supportive guidance.

Alternative methods for introducing CT have been explored over the last two decades. The "Computer Science Unplugged" movement has shown that unplugged activities—teaching computing concepts without computers—can successfully build intuition and engagement (Bell et al. 2009). By using games, puzzles, and physical activities, learners can experiment with computational structures in a low-stakes environment. Similarly, tangible resources such as cards, tiles, and worksheets have been used to teach sequencing, logic, and algorithms in ways that reduce dependence on technology infrastructure (Curzon et al. 2014). These approaches are especially valuable in contexts where computer access is limited, such as in under-resourced schools or outreach programmes (United Nations Educational, Scientific and Cultural Organization (UNESCO) 2023).

This chapter discusses two approaches to teaching programming and CT without the use of computers. The first is the **ProgBoard**, a printable tool that enables learners to explore sequence, selection, and iteration. The second is a **play-kit for first-year computing students**, developed to foster computational thinking through playful and tangible activities. The chapter then draws connections between these approaches and the wider challenges of accessibility and inclusion in programming education, concluding with a reflection on the potential of such methods for diverse learner groups.

9.2 Unplugged Programming with ProgBoard

The **ProgBoard**, developed as part of the "Think Like a Computer" initiative (Cutting 2025) and available from thinklikeacomputer.org[1], provides learners with a structured way to engage with programming fundamentals in a tangible form. It is a printable teaching tool designed to introduce the three fundamental programming constructs of **sequence**, **selection**, **iteration**, and **variables** all without requiring a computer.

The board itself consists of a grid on which tokens representing instructions can be placed. Problems are posed in the form of challenges, such as navigating a path, following rules, or producing repeated patterns. Learners construct solutions by arranging instruction tokens in order, creating simple algorithms that can be "executed" by following the board step by step.

[1] https://thinklikeacomputer.org

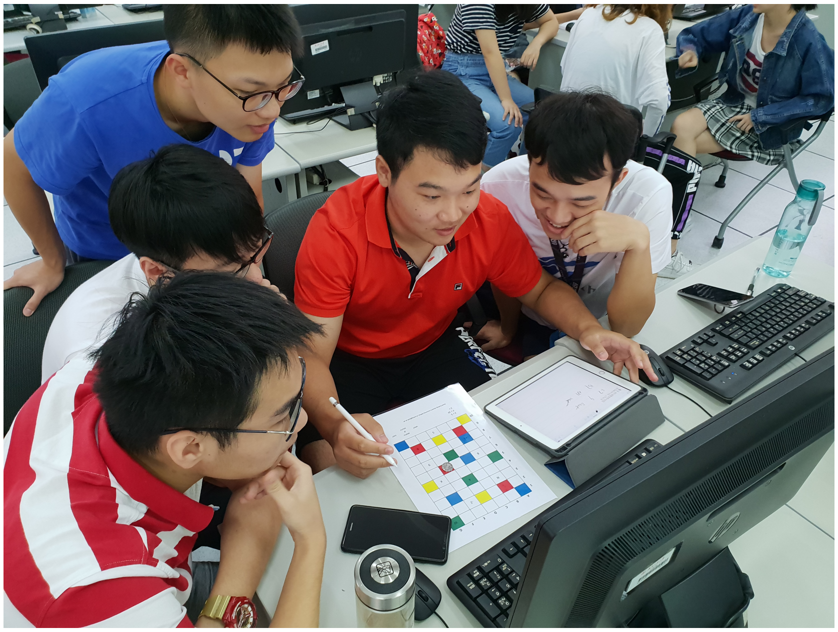

Figure 9.1: A ProgBoard being used to break the ice with during international programming teaching in China.

9.2.1 Sequence

Sequence is taught by requiring learners to arrange steps in a specific order. For example, if the task is to move an object from one side of the board to another, instructions such as “move forward”, “turn left”, and “pick up” must be placed in the correct sequence. Learners can then manually follow the sequence to check whether the task is achieved. This provides an embodied experience of algorithm design, reinforcing the importance of ordering in programming.

9.2.2 Selection

Selection is introduced by conditional markers. For instance, learners may encounter a task where an object must only be picked up “if it is red”. Tokens representing conditional statements are placed on the board, and learners must branch their instructions based on the condition. This tangible representation of branching logic helps learners to understand one of the most challenging concepts in programming: making decisions based on conditions.

9.2.3 Iteration

Iteration is represented by tokens that indicate repetition. For example, a loop marker may allow a sequence of steps to be repeated a specified number of times. Learners may be tasked with drawing a square by repeating a set of instructions four times. Through physically repeating steps, they grasp the efficiency and power of loops compared to writing out repeated instructions.

9.2.4 Variables

Variables are introduced in the form of counters ("keep count of how many…") initially and then as **named variables**, for example "x is three, move forward x squares and each time your counter is in a red square, add one to x". The use of variables in combination with selection for branching logic and iteration allows learners to see how simple algebra aligns with variables in programming and also introduced the concept of state.

9.2.5 Benefits of the ProgBoard

The ProgBoard reduces cognitive load by focusing on concepts rather than syntax. By externalising algorithms into a physical form, learners can see and manipulate their thought processes. Errors become visible as misplaced tokens or illogical sequences, which can be corrected collaboratively. This is in contrast to the intimidating error messages encountered in programming environments.

The tool also supports **collaborative learning**. Students can work in groups, discuss solutions, and challenge each other to design efficient algorithms. Peer explanation reinforces understanding, as learners must articulate the reasoning behind their token placements.

Research on unplugged activities has shown that such approaches are particularly effective for learners with limited prior exposure to computing (Denning 2017). By engaging learners in hands-on tasks, the ProgBoard provides a stepping stone into programming that lowers anxiety and builds confidence.

9.3 The Play-Kit Approach for First-Year Students

A second approach to programming without computers has been developed through the design of a **play-kit for incoming first-year university students**. This initiative, derived from the work undertaken at Maynooth University on their primary school CT workbook, aimed to introduce learners to computational thinking through playful, tangible interactions (Anderson et al. 2025) so is therefore widely applicable and not limited to students studying computer science.

The project emerged from the recognition that incoming students often have diverse levels of experience in computing. Some may have extensive programming backgrounds, while others may never have written a line of code. A one-size-fits-all approach risks alienating both groups.

The play-kit therefore sought to provide accessible, engaging activities that introduce CT concepts in a non-intimidating way.

9.3.1 Design Principles

The play-kit was co-designed by students and academic staff, drawing on resources originally developed for primary school learners but adapted to suit a university-level audience. Several guiding principles shaped its design:

1. **Accessibility** - reliance on colour was reduced to support learners with colour vision deficiencies.
2. **Visual instructions** - activities emphasised pictorial guidance rather than dense text, reducing language barriers.
3. **Playfulness** - tasks were framed as puzzles or games, encouraging creativity and engagement.
4. **Breadth of CT skills** - activities covered decomposition, algorithms, pattern recognition, logic, representation, and abstraction.

9.3.2 Examples of Activities

- **Bracelet Patterns**: Students extend and complete sequences of beads arranged in patterns, developing skills in pattern recognition, decomposition, and logic.
- **Code Breaker**: A symbolic substitution exercise where learners decode messages using simple ciphers. This fosters skills in representation and abstraction.
- **Escape the Grid**: A maze-based challenge where characters must move according to specified rules. This introduces conditional logic and algorithmic pathfinding.
- **Castles and Wyverns**: A dice-based game where outcomes depend on nested conditions, encouraging learners to apply probabilistic reasoning and selection.
- **Marble Race**: Learners simulate the timing of marbles released into tracks, exploring concurrency, sequencing, and time-based reasoning.

These activities allow learners to explore computational concepts in a playful environment. Unlike programming assignments that require precise syntax, the play-kit enables safe exploration. Mistakes are opportunities for discussion and learning rather than points lost.

9.3.3 Preliminary Evaluation

Early feedback from workshops suggested that the play-kit was effective in lowering barriers for learners who felt intimidated by programming. Students reported increased confidence, greater willingness to collaborate, and enjoyment of the playful format (Anderson et al. 2025). The tangible, hands-on nature of the activities was particularly valued, as it contrasted with the abstractness of code.

9.4 Discussion

The two approaches explored in this chapter—the ProgBoard and the play-kit demonstrate the potential of introducing computational thinking (CT) and programming concepts without direct reliance on computers. While they differ in context, design, and target audience, they share a common aim: to make programming more approachable by lowering barriers and foregrounding concepts before syntax. By situating programming in a tangible and playful space, both approaches challenge the assumption that programming must begin with a computer and instead suggest that thinking skills can, and perhaps should, be nurtured in more accessible ways.

One of the most significant contributions of unplugged approaches is the way they reduce **cognitive load** for beginners. Research has consistently shown that novices often struggle with the dual burden of understanding abstract programming concepts and mastering the mechanics of a programming language at the same time (Sweller 1988). For instance, a student may conceptually understand that a loop repeats a set of actions, but may be unable to express this understanding in Python, Java, or C without encountering frustrating syntax errors. By externalising programming logic into physical forms, as seen in the ProgBoard tokens or play-kit puzzles, learners can focus on what the computer is being asked to do, rather than how the instructions are formally expressed. This separation of concerns provides a clearer path to mastery: concepts first, formal languages later.

Equally important is the potential for these activities to support **diverse learner groups**. Traditional programming environments tend to privilege students who are already confident with computers or who have prior experience in coding. This creates inequities, with learners from non-computing backgrounds often left behind. Lifelong learners, interdisciplinary students, and those with limited digital literacy can find the initial learning curve intimidating, reinforcing the stereotype that programming is only for “tech experts.” By contrast, both the ProgBoard and the play-kit offer accessible, low-barrier entry points where learners can engage in computational problem-solving without needing technical fluency. This is especially relevant in outreach contexts, professional development programmes, and in regions where reliable access to computers is limited.

The **affective dimension of learning** should not be overlooked. Fear of failure is a well-documented barrier in programming education, where small mistakes in syntax can lead to discouraging error messages and wasted time (Watson and Li 2014). In unplugged contexts, however, mistakes become opportunities for dialogue and playful exploration. If a sequence of tokens fails to solve a ProgBoard puzzle, learners can quickly rearrange the steps and try again. If a group misinterprets the rules of a play-kit game, the result is not an intimidating compiler error but a chance to reflect, discuss, and iterate. This transformation of errors into constructive learning moments fosters resilience and helps to cultivate a growth mindset around programming.

Furthermore, the **playful nature** of these activities contributes to learner motivation and engagement. Play has long been recognised as a powerful mode of learning (Papert 2020a; Whitton 2022). It encourages experimentation, risk-taking, and persistence, qualities that are central to effective problem-solving in computing. By embedding CT tasks into puzzles, games, and tangible challenges, the ProgBoard and play-kit avoid framing programming as a purely technical skill. Instead, they emphasise creativity, exploration, and fun—qualities that can be especially important for engaging students who might not initially see themselves as "coders." For example, the fantasy-themed "Castles and Wyverns" dice game from the play-kit illustrates how algorithmic logic can be learned through an imaginative context, rather than an abstract lecture on conditional statements.

The **transferability of skills** developed in these unplugged settings is also crucial. While the activities do not produce working code, they cultivate habits of mind—such as decomposition, abstraction, and algorithmic reasoning—that can be readily applied in formal programming environments. In this way, unplugged activities function as a **bridge**. Once learners have gained confidence with the underlying concepts, they are better equipped to face the complexity of programming languages and development environments. This scaffolding effect has been linked to improved retention and success rates in introductory programming courses (Luxton-Reilly et al. 2018).

Finally, both approaches point to the **broader potential of programming without computers**. Their application need not be limited to early stages of computer science education. They could be adapted for interdisciplinary teaching, allowing students in fields as varied as biology, business, or the arts to engage with computational thinking in a way that feels relevant to their domain. They could also be used in informal education settings—libraries, community workshops, or online learning platforms—to broaden participation in computing. For educators, unplugged methods provide flexible, low-cost tools that can be adapted to different curricula, learner profiles, and institutional contexts.

Taken together, these observations suggest that unplugged approaches should not be viewed merely as supplementary activities or temporary scaffolds, but as integral components of a broader strategy to democratise computational education. By making programming more tangible, playful, and inclusive, they help dismantle barriers that have historically excluded many learners from engaging with computing. In doing so, they lay the groundwork for more diverse, resilient, and conceptually grounded cohorts of future programmers.

9.5 Conclusion

Programming without computers represents an effective and inclusive approach to developing computational thinking skills. By focusing on concepts rather than syntax, learners are able to engage with programming fundamentals in an accessible, playful, and low-stakes environment.

The ProgBoard demonstrates how simple, printable resources can teach sequence, selection, and iteration. The play-kit for first-year students illustrates how playful, tangible activities can introduce decomposition, algorithms, logic, and abstraction in ways that reduce anxiety and increase motivation.

Together, these approaches highlight the value of unplugged activities as a means of lowering barriers, supporting diverse learners, and broadening participation in computing. They serve as stepping stones to computer-based programming, ensuring that learners enter digital environments with confidence and conceptual clarity.

It is concluded that the continued development and evaluation of unplugged programming approaches has significant potential for education at all levels. By reimagining how programming is introduced, educators can make computational thinking accessible to all learners, regardless of background or prior experience.

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

10.1 Introduction

This chapter discusses the Centre for Data, Culture & Society's (CDCS) training initiative, which aims to integrate computational methods into humanities research. CDCS is committed to fostering methodological innovation and advancing digital research skills within the academic community. It provides targeted support and expertise to both individual researchers and groups, enhancing their capacity to employ digital techniques in the arts, humanities, and social sciences. Importantly, the initiative bridges the gap between technology and traditional research methods, preparing scholars to adeptly incorporate digital tools into their work. The CDCS training programme is distinguished by its interdisciplinary and peer-led educational model, which serves researchers at different stages of their career and with very heterogenous levels of digital literacy. The programme fosters a shared language across disciplines, enabling scholars from non-technical backgrounds to embrace coding-based methods confidently. A natural outcome of this approach is that it enables us to observe how the newly acquired shared language facilitates collaboration and interdisciplinary work, effectively breaking down traditional disciplinary barriers. This aspect is particularly crucial in enabling researchers trained in non-technical disciplines to confidently navigate and employ complex digital tools in their work.

10.2 The Centre for Data, Culture & Society (CDCS)

Today's researchers work in a rapidly changing technological environment, tackling complex global challenges: the ability to work across disciplines, scales, and sectors is increasingly important. In this context many researchers are embracing data-led approaches and there is an increase in demand for training in the kinds of applied methods that would traditionally have been taught only in computer science and STEM disciplines through coding-based approaches. Scholars in the arts and humanities, who bring invaluable critical, creative and ethical perspectives to interdisciplinary projects, face specific challenges in acquiring these technical skills: barriers include confidence, grounding assumptions, language and a lack of tailored skills development opportunities. The inherent complexity of computational methodologies, along with unfamiliar technical language and processes, can prove a significant stumbling block to researchers whose time and resource is limited. Empowering such researchers to engage and explore the value of these approaches supports not only more effective collaboration, but also more nuanced, reflective and ground-breaking interdisciplinary projects.

The Centre for Data, Culture & Society (CDCS)[1] at the University of Edinburgh[2] was founded with a mission to inspire and support scholars across the arts, humanities and social sciences to explore, navigate and utilize the potential of data-led and computational methods. To build capacity and support for interdisciplinary digital research projects, and to support the sharing of relevant knowledge, methods and resources between subject areas and schools, it was deliberately positioned outside of academic departments: support from a cross-school academic advisory board ensures that all of the research communities within the College of Arts, Humanities and Social Sciences can feed into planning and prioritization. Through a range of technical services, CDCS offers a space for experimentation and tries to catalyze innovation, fostering a culture of peer-collaboration and knowledge sharing. It provides tailored and flexible support including bespoke technical support and advice, prototyping sandpits, data clinics and opportunities to develop communities of practice. This collaborative ecosystem is further enriched by regular networking events and workshops that provide flexible opportunities for professional development and interdisciplinary engagement. The Centre is staffed by a professional services team that includes a director, training manager, and events coordinator as well as training fellows and research software engineers.

10.3 The CDCS training programme

The CDCS training programme is a key part of the Centre's offer and has been developed over the last six years to provide participants with practical support in applying computational methods to their projects. It is designed to be inclusive, accessible, and practical, and offers a variety of formats to accommodate diverse learning preferences, ranging from workshops and seminars to intensive boot camps, self-paced learning materials and online courses. 'Digital Method of the Month' sessions, for example, provide beginner-friendly introductions into a specific approach or technique.[6] These hour-long sessions are designed to give researchers a practical sense of what each method involves before committing significant time to it. They create a welcoming, entry-level space for exploring new approaches, with frank discussion of tools, challenges, and the realities of applying methods in practice.

[1] https://www.cdcs.ed.ac.uk/

[2] Founded in 2019, CDCS is a strategic initiative of the College of Arts, Humanities and Social Sciences at the University of Edinburgh. Since 2021 it has been part of the Edinburgh Futures Institute[3], one of five innovation hubs established through the Data-Driven Innovation Programme[4] funded by the Edinburgh and South East Scotland City Region Deal[5]. Alongside its training programme, CDCS offers events, technical services and infrastructure for data-led research.

[6] For each meeting, we also prepare a Markdown file[7] that provides attendees with starting notes to support their work on a specific method.

A foundational principle of the training programme is that it is ‘by researchers, for researchers’ and delivered by a cohort of Training Fellows (TFs), leveraging peer-led instruction to build competencies within a supportive community. The fellowship program recruits doctoral candidates and early-career researchers who use computational methods in their projects.[8] It offers them an opportunity to deliver courses, teach and mentor their peers, thus fostering an environment of reciprocal learning and reinforcing their own understanding of the subject matter. This peer-led structure serves to nurture networks of support and collaboration. A key example is the Centre’s Bring Your Own Data (BYOD) sessions, which enable students to bring their work into the learning environment, discuss their processes and problems and troubleshoot together. This allows participants to get hands-on experience and supports learners in translating their new skills to their own research environments, thus reducing barriers to implementation and encouraging ongoing usage of computational methods.

One of the main challenges in teaching computational methods to researchers in the Humanities and Social Sciences is the technological language barrier. Many participants enter workshops without a shared vocabulary for describing what they are trying to achieve through the application of code-based tools. The Centre tackles this by intentionally focusing on building a shared terminology that helps attendees build up their understanding while mapping their research needs against specific computational methods. Instructors provide clear definitions of key terms, introduce foundational structures, and carefully explain the “grammar” and syntax of computational processes. This approach equips participants with language that they can use to continue learning independently – as we always tell learners, half of the battle is knowing what words to google!

Concepts are made more accessible through metaphors and examples drawn from the humanities and social sciences. For instance, the CDCS Training Manager Lucia Michielin, often introduces the process of building working code chunks by way of a comparison to translating

[8] The Training Fellows are recruited annually (with the possibility of extending their contract) through an open call to current PhDs and ECRs, and through a standard interview panel. Once appointed, they are offered a Guaranteed Hours contract.

This provides flexibility, assuring them sufficient hours to design and deliver one training course and leaving open the possibility of taking on more work, if they have capacity and interest. How many hours each TF works every year can vary considerably and is dependent on programming, availability, skills, and budget. Once they start their contract, they receive a general induction covering our aims and delivery methods, past training courses and available materials. After that they work with the Training Manager to design the programme.

Each year there is a combination of returning and new Training Fellows, and during the first semester the new TFs normally start as helpers in courses led by more experienced Training Fellows or by running courses where the material has already been consolidated and just needs a simple refresh and update. During this phase, they learn how to provide feedback on material delivered by others, and they familiarise themselves with different methods of delivery. Thereafter, more often in the second semester, they are invited to submit ideas for new training material or more consistent revisions of existing training material. At each stage, they receive and give feedback on materials and delivery methods, both from their peers and from the Training Manager (we use GitHub for our materials as it supports this constant review process).

The pedagogical progress of the Training Fellows — from helping in a class led by someone else to producing brand new training material—is usually dependent on their previous experience and capacity, but by the end of their contract, they will all have experience in developing new or extensively revised training material.

dead languages such as Ancient Greek or Latin. This analogy is grounded in real learning experiences as it relates to her experience as a Classics PhD, trying to find a way to connect unfamiliar computational methods to research tasks she already understood. Training Fellows are encouraged to foreground their own learning experiences and the analogies they have found helpful, which gives trainees insight not only into the method at hand but also into others' learning processes and how they can be adapted to different contexts.

Lessons pay close attention to vocabulary, avoid unnecessary jargon, and focus on core principles, ensuring that no prior computational knowledge is assumed. Participants are encouraged to discuss, consolidate, and actively apply new terminology, reinforcing understanding and enabling them to articulate complex ideas in computational terms. The learning process is framed as analogous to acquiring a new language: mastering basic structures first, then building fluency through repeated practice and application.

Given the international dimension of researchers at the University of Edinburgh, this language barrier can sometimes extend to natural language itself. International students and researchers may struggle to find the right terminology to articulate their needs, particularly when confronted with frustratingly obscure error messages. A helpful strategy is to encourage them to set their coding interface to their native language. This forces them to interpret and translate the error into English when explaining it to the instructor. This mental process of navigating between languages not only clarifies the issue for the instructor but also helps the learner develop a deeper understanding of the underlying computational problem and strengthens their ability to articulate technical challenges in a second language.

By establishing a shared language, the Centre also promotes interdisciplinary collaboration. Researchers from different backgrounds gain confidence in approaching computational problems, bridging gaps between disciplinary cultures. This not only enhances the immediate learning experience but fosters sustained engagement with computational methods: Participants are more likely to translate their new skills into their own work, reducing barriers to implementation, and encouraging ongoing use their learning. As a shared language is developed, it also provides a common framework for exploring questions and problems. A good example of this is the success of the "Data Surgeries" sessions, which provide one-to-one meetings where researchers receive tailored advice on applying digital and data-driven methods to their projects. These hour-long consultations are led by our Training Fellows, who often come from different disciplinary backgrounds than the researchers they are assisting. Fellows help with data processing, troubleshooting, and guide participants toward additional resources or further training.

10.4 Open training materials

Alongside the Training Fellow-led programme of courses and workshops, the Centre focuses on reinforcing an interdisciplinary and peer-led approach through a range of formats and materials. Since July 2020, the Centre for Data, Culture and Society has been a member of the *Programming Historian*[9] Institutional Partner Programme, a partnership that reflects our sustained commitment to open, peer-reviewed, and sustainable training resources. In practice, many *Programming Historian* tutorials have been incorporated into CDCS activities: they are used in *Masterclasses* (ad hoc deep dives into innovative computational methods for research) and in *Silent Discos* (asynchronous online workshops; see Cooling et al. (2026) in this volume).

The tutorials also underpin the “further reading” sections of the Centre’s structured *Training Pathways*[10], guiding learners towards high-quality, freely accessible materials that extend beyond the workshops themselves. Covering diverse topics such as data visualisation, statistical analysis, text analysis, sentiment analysis, and working with 3D Data, these Training Pathways are designed to provide structured, accessible guidance through a wide range of digital research methods. They are aimed at beginners and offer a scaffolded approach that helps learners identify which skills to acquire first and how to progress: each pathway highlights essential steps, tools, and concepts, while providing guidance on potential challenges and pointers to areas for further exploration. Rather than presenting methods in isolation, the pathways emphasize the connection between methods and underlying skills, highlighting often overlooked but crucial practices. For example, they make clear that even the most polished data visualisation is meaningless if the underlying dataset has not been rigorously collected and carefully cleaned. By drawing attention to these foundational steps, the pathways reveal the “hidden” work that underpins robust and reliable research outcomes, and, by combining practical instructions with curated “further reading,” they give learners confidence and direction, enabling them to approach computational methods incrementally rather than feeling overwhelmed.

Since its launch, the Centre has produced a substantial body of training materials designed to be open, adaptable, and widely accessible to any discipline. These resources are collaboratively developed by a network of researchers from diverse research backgrounds, ensuring that all materials are proofed and checked to ensure they are jargon-free and speak effectively to a general scholarly audience. All materials are shared openly via GitHub repositories[11] under a CC-BY 4.0 license, ranging from markdown help pages to PowerPoint presentations and notebooks (mostly R or Python based). This open model not only ensures that the resources can be freely accessed and reused but also encourages continuous refinement and improvement based on user feedback and emerging research needs.

10.5 Our impact

Overall, CDCS plays a pivotal role in removing barriers that have traditionally hindered the use of computational methods in various disciplines. This flexible approach has proven successful, with around 600 to 700 researchers per year benefiting from the Centre’s work. By

[9] https://programminghistorian.org/en/

[10] https://www.cdcs.ed.ac.uk/training/training-pathways

[11] https://github.com/DCS-training

emphasizing interdisciplinarity, peer-support, flexible formats, and explicitly showing learning processes, the Centre is effectively transforming how researchers in the arts, humanities and social sciences engage with digital and computational technologies. This approach not only enhances individual research capacity but also contributes to the broader academic and societal understanding of data, culture, and society, leading to innovative solutions to complex global challenges. As computational methods continue to weave into the fabric of research across disciplines, initiatives like those at CDCS will be indispensable in preparing researchers to navigate and lead in this new digital era.

10.6 *Appendix 1: Digital Method of the Month (DMM) session in detail*

DMM sessions are informal meetings to create a safe space to freely discuss the practicalities of learning and implementing a new digital skill in research. Each month a method is selected, and the meeting itself is a 1-hour practical discussion on what it takes to learn and master the method. During the meeting, we try to answer the following questions:

- What this method is really about?
- How much time will it take to get the basics?
- What are the software options available?
- What are the most common pitfalls?
- Where can you find more info on the subject?

A typical schedule is as follows:

- 00:00-00:10 Housekeeping and intro
- 00:10-00:25 Round of Presentation of the attendees (to collect specific interest and hurdles of the attendees)
- 00:25 –00:50 Discussion using a Markdown document
- 00:50-01:00 Discussion on available resources for learning, way forwards and wrapping up

Example: Digital Method of the Month on Working with GIS and Geographical Datasets

Learning objectives:

- Become familiar with the main concepts of Geospatial Data and GIS
- Understand the easiest pitfalls new practitioners can run into when starting working with these tools

Materials provided:

- Housekeeping slides with rules of the DMM
- Markdown file with major points of discussion written out with hyperlinks to further resources (all previously run DMM Markdown files are available online[a])

Main discussion points:

- Concepts and jargon that practitioners need to familiarize with — e.g. difference between vectors and raster, and practical aspects of working with different data sources, standards of data (like shapefiles, GeoPackages, GeoTiff, etc.), downloading data vs. connecting to OSM and WMS services, etc.
- Time frame for learning and important theoretical topics — e.g. the importance to fully understand how reference systems work and what are the consequences of picking the wrong reference system
- Easy pitfalls for newbies — e.g. propriety vs open source extensions, information tree organization, etc.
- Future possible issues — thorny issues that learners will encounter down the lines once they will move to more intermediate and advanced techniques (e.g. implementation with coding environments, working collaboratively, managing 3D data)
- Main software available, with pros and cons of the main options
- List of helpful websites/blogs on where to ask for help
- Resources to learn GIS, both as self-learner and within the Centre and the University in general

[a] https://github.com/DCS-training/Digital-Method-of-the-Month/tree/main/DMM%20Docs

11 Learning Together Across Modes: Online and On-site Pair Programming in a Hybrid / Fusion Course

11.1 Introduction

“Sarah, can you scroll down a bit?” The voice appeared suddenly in Sophie’s headphones, causing her to jump slightly in her chair. Just moments before, she’d been chatting face-to-face with Dr Llewellyn about Python syntax errors. Now her disembodied voice was helping her online partner navigate their shared code, while she remained physically present in the classroom just three feet away. Welcome to fusion teaching, where instructors become friendly ghosts and the boundaries between digital and physical learning dissolve in unexpectedly delightful ways.

In this chapter, we want to share our experience of developing and teaching an experimental and innovative pair programming course.[1] This course is non-traditional; we teach text mining to master’s level students in fusion mode (occurring in a highly technologised classroom designed for teaching online and on-site students as a single cohort to involve all students as fully as possible)[2] in a two-day intensive session. We knew this would be challenging, but we also knew that this would be a great learning experience and a chance to do things differently. We present here an honest reflection of the trials, tribulations, and wins in what was ultimately a rewarding and satisfying experience. We didn’t do everything perfectly, but we learned a lot, and we believe the students did too. Please read on and hopefully enjoy hearing about our experience.

Pair programming (Orzechowski et al. 2026; Desvages et al. 2026) is central to our course, but pairing students effectively in a fusion setting presents its challenges. It was important to us for our entire cohort to feel a sense of parity; this could not be an in-person course with people watching online. We wanted to develop fluidity between the modes. We worked hard at this and broke down the barrier between the room and online. We created a classroom in which both teachers and students can be vulnerable and open, so that we can grow from each other’s mistakes, and communicate with respect.

[1] At the University of Edinburgh a “course” is usually a block of teaching (5-10 weeks) on a subject with assessment at the end, which awards academic credits. In your institution this might be called differently, e.g. class, subject, module or topic.

[2] See QAA Scotland - Examples of good practice[3]

When we first designed and ran *Text Mining for Social Research*[4], the Edinburgh Futures Institute's first fusion course at the University of Edinburgh, we knew we were stepping into uncharted territory (Alex et al. 2021). Fusion courses integrate both in-person and online students, requiring thoughtful interaction strategies to create a cohesive learning experience. While hybrid teaching models have been explored extensively (Beatty 2019; Raes et al. 2020) and so has the value of peer instruction (Porter et al. 2011), our fusion approach to intensive pair programming represents a novel application of these principles to collaborative coding education. With a team of four (three lecturers and one teaching assistant), we had the flexibility to experiment with innovative teaching strategies that bridged the gap between online and on-site students. This chapter presents our experiences and lessons learned when experimenting with three key interaction methods: *Negotiating Fusion-Pairs*, *the Ghost Helper Effect* and *Micro-interactions*.

11.2 Making Fusion Pairs Work

A reflection on what needs to be considered when pair programming in a fusion classroom

Pair programming is central to our course (Williams et al. 2000; Hannay et al. 2009; Hanks et al. 2011), but pairing students effectively in a fusion setting presented its challenges. We experimented with assigning students in same-mode pairs (on-site-on-site or online-online) and mixed-mode pairs (on-site-online), adjusting pairs every hour to help students feel part of a single, integrated cohort. Each new pairing required students to renegotiate key aspects of their collaboration: how they worked together, who took the lead, and how they interacted with instructors. This constant reshuffling reinforced the sense of a shared learning experience across both modes.

We encountered several challenges in this type of paired learning approach (audio issues, cognitive load, context switching). This could especially affect students when accessibility issues or other special circumstances are compounded or clash with the learning environment.

While students generally preferred mixed-mode pairs (see Figure 11.2) and reported feeling more connected to the broader cohort, this arrangement proved exhausting to both online and on-site participants. The background noise in the classroom, combined with the additional cognitive load students experienced as a result of this noise, can leave them feeling drained by the end of a session.

The size and acoustics of the classroom are key factors to consider when using pair programming in teaching. Background noise not only causes "Zoom fatigue" for online students (Bailenson 2021), especially in longer events like ours, but also affects the on-site students who wear headphones to connect to their mixed-mode buddy. Such overload issues disproportionately

[4] This was a two-day intensive bootcamp course for 30 postgraduate students (15 on-site, 15 online). Each day consisted of four 1.5-hour-long sessions, with each session starting with a 15-minute introduction, an hour of coding in pairs and a 15-minute Q&A session.

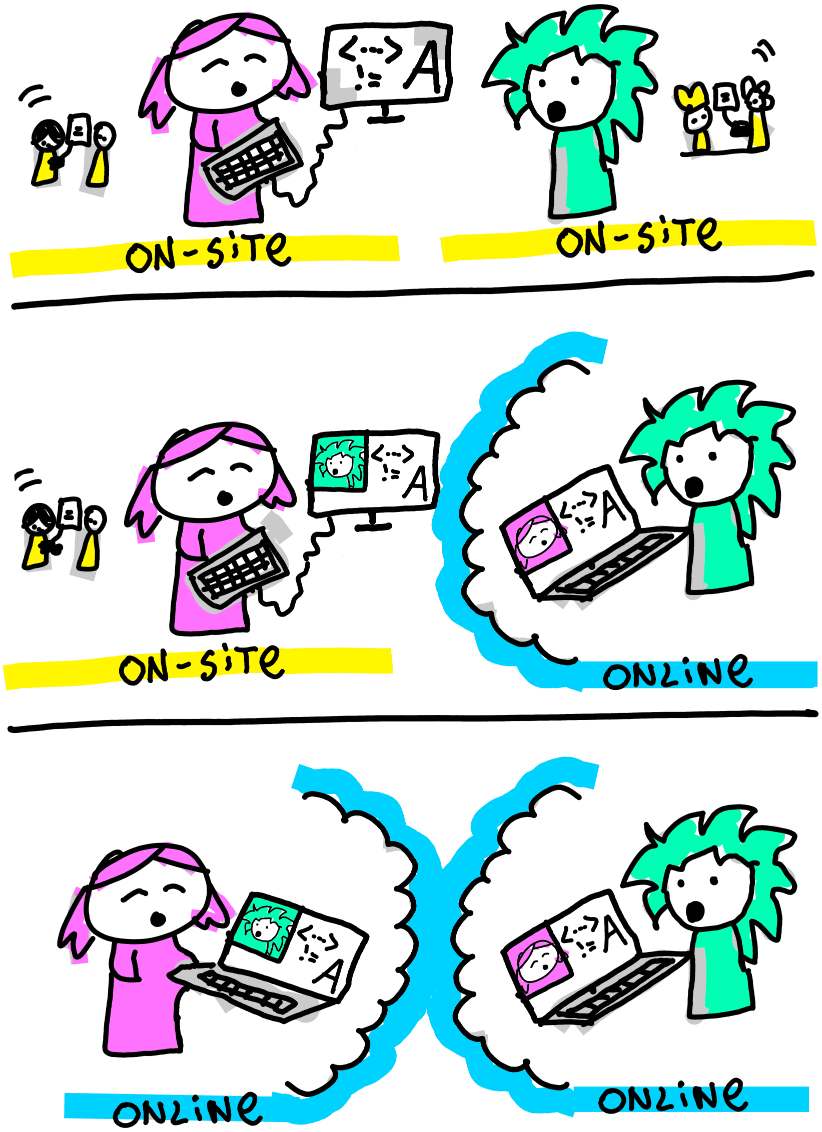

Figure 11.1: Different modes of paired students working together on programming exercises.

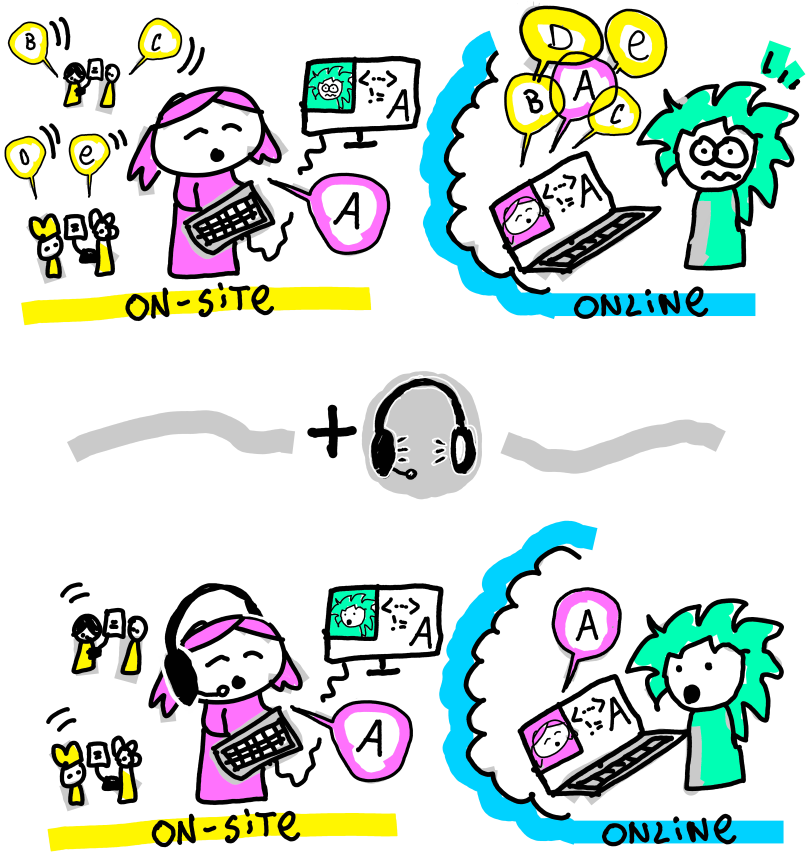

Figure 11.2: Mixed-mode pair programming in action with noise in background.

affect neurodiverse learners who may struggle with auditory processing or filtering competing stimuli. In our case, additional room facilities or even a quiet area outside of the classroom proved handy for such interactions. As a teaching team, we hadn't originally anticipated this need. We are known to improvise and have once used a "broom cupboard" as an overflow space to support mixed-mode pairs.

Unlike typical online interactions, where participants can selectively mute themselves, paired programming also demands constant verbal communication as students alternate between navigator and driver roles throughout exercises. This requirement for continuous dialogue means that small audio cues (*"aha," "umm"*) become important elements that would make the experience worse if they were missing. The downside of not muting ourselves is that background noises cannot be avoided.

As teachers, we had to learn what the experience of an online participant is. When joining students online, we noticed how important the camera angles are. Online students like to see teachers close up but also what's happening in the classroom. When teaching hearing-impaired students online, they like to see the face of the lecturer close up for lip reading. When looking at code on a screen, the font size needs to be sufficiently big, and it is important to scroll in a controlled way, as otherwise it becomes difficult for the partner to follow. Often, the first few minutes for each pair's collaboration were taken up by making all those audio/visual adjustments, and then they were ready to go.

With several mixed-mode pairs in one room, there might also be a feedback loop (which sounds like a high-pitched squeak when a microphone picks up the sound from the speakers). This is something that needs to be managed carefully, and, again, additional physical space will help with that. In an hour-long session, such adjustments can take up another few minutes to sort out.

More broadly related to fusion teaching, addressing and talking to online students explicitly during the introduction of the course and throughout, and not just the ones, is vital to make them feel part of the group, as it is very easy to forget about them otherwise. Having personally participated in hybrid courses that prioritised those in the physical classroom, the online experience can feel isolating. Inertia and gravity pull teachers towards excluding online participants, if there are active efforts and structures to include them. For our text mining course, we rehearsed all possible scenarios of interacting with on-site and online students in the bigger group or in pairs (e.g. how do I interact with a student pair that is mixed-mode, on-site-only or online-only) and really thought carefully about what is important in each situation. We use an advocate for the online students in the room, a person with specific responsibility, who could be another teacher, a teaching assistant, or a student, to draw attention to any comments or raised hands from an online student that the teacher may have missed. This has the dual effect of including the online students in that moment but also reminding the teacher to check in the future. Making space to ask the online students for their comments is important; they become more active participants, and the students in the room feel like they are part of the cohort.

After several years of practice teaching using pair programming in this and similar courses, we are tending more towards online-only and on-site-only pairs, given that some of the challenges involved (e.g. classroom setup) are out of our control and are not something we can easily adjust during the course. In some ways, this feels like a failure; we really wanted to completely cross that boundary, but if this makes it harder for the students, we must adapt. We found that running the mixed-mode sessions only in the morning, when everyone is fresh, allows us to feel like one group and reduces the likelihood of Zoom fatigue later in the day.

11.2.1 Ghost Helper Effect and the Fluid Divide

A discussion on a fluid divide and not siloing students into two groups

In a traditional classroom, students can simply raise a hand and ask the teacher for help. However, in a fusion course environment, mixed-mode pairs introduced a new dynamic. We quickly learned that if online students needed help, support also had to come online. Otherwise, teachers ended up only communicating with people in the room. To address this, all teaching staff were online during pair programming sessions, creating what we called the "ghost helper effect". This allowed us to speak to students in both modes simultaneously, whether through direct conversation in the room or via webcams and headsets to ensure an equal experience.

This led to a surprising phenomenon for on-site students in mixed-mode pairs, suddenly hearing a voice in their headphones from one of the teaching team they had just interacted with in person a few minutes earlier, like a friendly ghost. At first, this unexpected presence was disorientating, diverting attention and requiring a cognitive shift (see Figure 11.3). To normalise the experience, we consciously used online helpers for all mixed-mode pair work. While initially some students were spooked by the fluidity of the "ghost" seeming to be everywhere at once, they quickly found this both friendly and helpful.

In taking this approach, it was important for students to feel a sense of parity in how the course was being delivered and to develop a fluidity between online and on-site. With staff being online and visible through their own cameras, they could speak to both groups directly through the same method. Rather than speaking to an online student through an on-site partner's webcam or relaying information to be passed on, this ensured that all information was delivered simultaneously and created a sense of fluidity between students and staff.

This was strengthened by staff at times leaving the main classroom and speaking to students in different locations, for example, a smaller classroom. They effectively became 'online' as well, working to break down the divide between the two modes, while still communicating with students together. In blurring the boundaries, neither group felt staff were giving preference to one side or the other, and the "ghost helper" approach allowed us to support students seamlessly regardless of location.

For students, this fluidity created a sense of seamlessness between the two media, and while much thought went into facilitating this, for the students, it felt effortless. Some even shifted

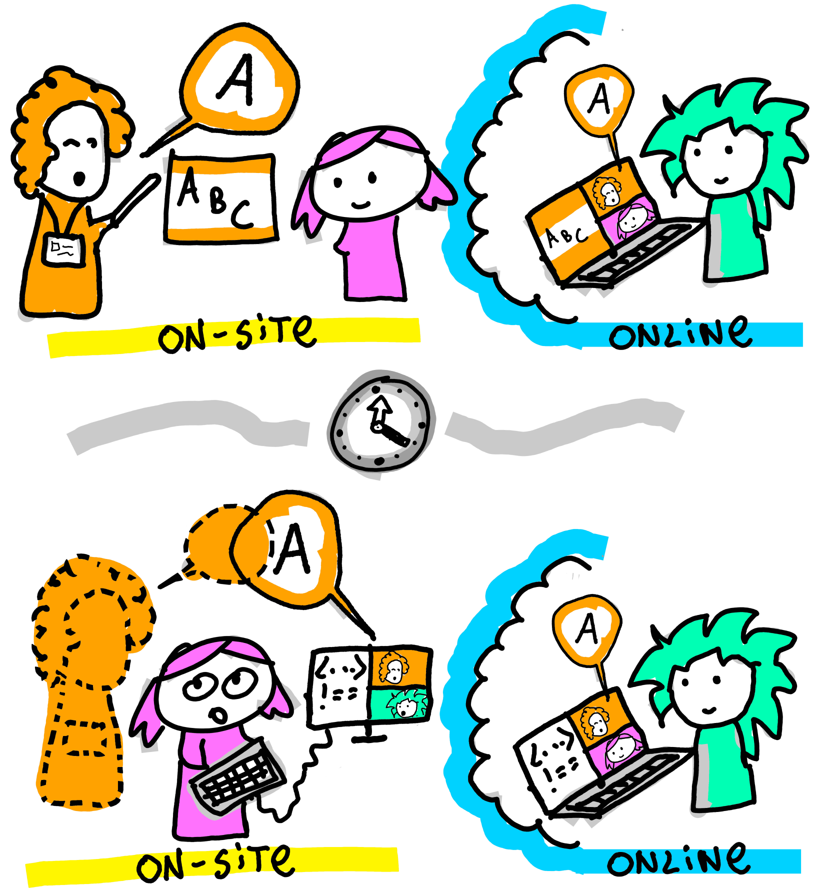

Figure 11.3: The ghost helper effect.

between modes during the course, starting the day online before coming on-site or the other way around. Allowing this flexibility showed that students felt at ease in either mode and that, while both had specific issues, neither was better than the other. For instance, at one point, electrical issues on-site meant that students in the room were unable to access their workbooks, while those online were unaffected. Therefore, while both modes could present challenges, in creating flexibility between them, students were able to adapt with little disruption.

11.3 Micro-interactions

Micro-interactions are how we bring the learning into the wider classroom

When you teach a course in an intensive mode, you are all together for two very busy days. This helps with a feeling of connection and that we are all in this together. It exacerbates feelings of joy but also of frustration. Learning happens in the midst of these feelings, and we know we need to connect the course with those small moments of joy and frustration and allow them to be voiced. We wanted to encourage, celebrate, and capture these moments of learning and cohort camaraderie, and quickly fix mistakes we were making and habits that were annoying.

Below we will explain strategies we employed to make the classroom a place of sharing, exchange and growing together. Feedback was gathered not only for quality control, as with the typical "How did you like the course?" questions, but rather to continuously synchronise with the energy and mood of all the students on the course; we call this 'relentless feedback'. It has to be quick and fun and not distract the class from the core learning. We gamified these small moments of interaction, with methods for ensuring everyone has an equal and anonymous voice, avoiding some common techniques which can be quite exclusionary, such as when a teacher picks the student who raised their hand first, it is often the same student. We asked students to put crosses on a whiteboard of questions to show where their mood was, and we asked them to fire answers to verbal questions in the chat function of the online room; we call this a 'Chat Blast'. If a student picked up a mistake in the code, we asked them to circulate it to the room. If they had a good comment or insight during their learning, we asked them to bring it back up in the discussion sessions when we came back together as a whole group after pair programming. We adapted sessions when people were frustrated or tired, we voiced that these feelings were valid and uncomfortable but can be part of learning, and we upped our energy when the energy in the room started to fade. We created a classroom in which both teachers and students can be vulnerable and open, so that we can grow from each other's mistakes, and communicate with respect (amplification of learning).

11.3.1 Relentless Feedback

We wanted to gather feedback as we went to have a quick and real-time reading on how the course is going and to be able to adjust our approach if needed. There is a trade-off between asking feedback on the spot or later: asking during the learning process can be quite disruptive, while if we ask later, students might not take the time to provide the feedback in the first place or might not remember how they felt earlier. This is especially important if we care about granularity and would like to know if, for example, a particular part of the course needs to be improved.

We envisaged a two-way feedback channel, where students say how they are feeling, but they also see how everyone else in the room is doing. We were also aware that anonymity will lead to more honest and useful answers. Another part was that we wanted it to be playful and not very constrained by the digital tool, where the little artefacts like handwriting, colour choice or exact position on the screen can convey personality and individuality.

We needed a quick and unobtrusive way to gather feedback, which meant that we could use it multiple times during the day. We also wanted to ask meaningful questions, which will help us guide the course delivery (not just "Do you like it") since we were aware of the complex interaction between whether a course is 'hard' and 'useful' (where 'hard' is fine, as long as it is 'useful').

We wondered whether we could create a feedback mechanism which is easier to complete than to dismiss/avoid, sort of like 'accept cookies' popups on websites make it easier to accept than to customise. So, we created a multiple-slider feedback mechanism, where students grade on a Likert scale multiple aspects of their learning experience, and all results are fully open but anonymous.

For our fusion course, we could not find a tool which would deliver exactly what we needed at the time, so we used a shared digital whiteboard. The whiteboard would always start with several empty scales, and each student would be invited to type the letter X in the appropriate place on the whiteboard (see Figure 11.4). Since we gathered this feedback every 1-2 hours of the course (after each badge), we had very granular data about the two days of our course. In the animation below, you can see the way the mood in the room changed as the course progressed, which was very useful for us to see and respond to.

For completeness, this technique is not limited only to online or hybrid environments, as we also tried it on in-person bootcamp courses as a means of gathering and sharing feedback in the room (Figure 11.5).

11.3.2 Chat Blast and Other Micro-contributions

Following the same philosophy as in the feedback above, we wanted to give students opportunities to check in with their own learning and the group and provide constructive feedback to the teaching team. Throughout the course, there were moments where we asked them to contribute questions that we would then discuss in the session after each pair programming exercise. This enabled the entire cohort to benefit from the discussion and answers provided. We also ensured

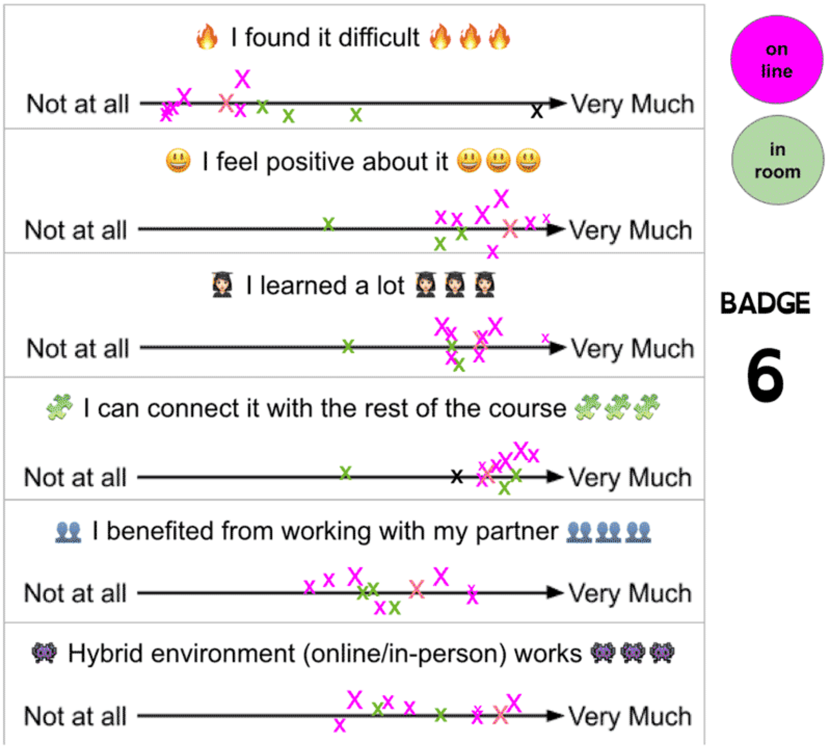

Figure 11.4: Feedback collected at the end of each badge.

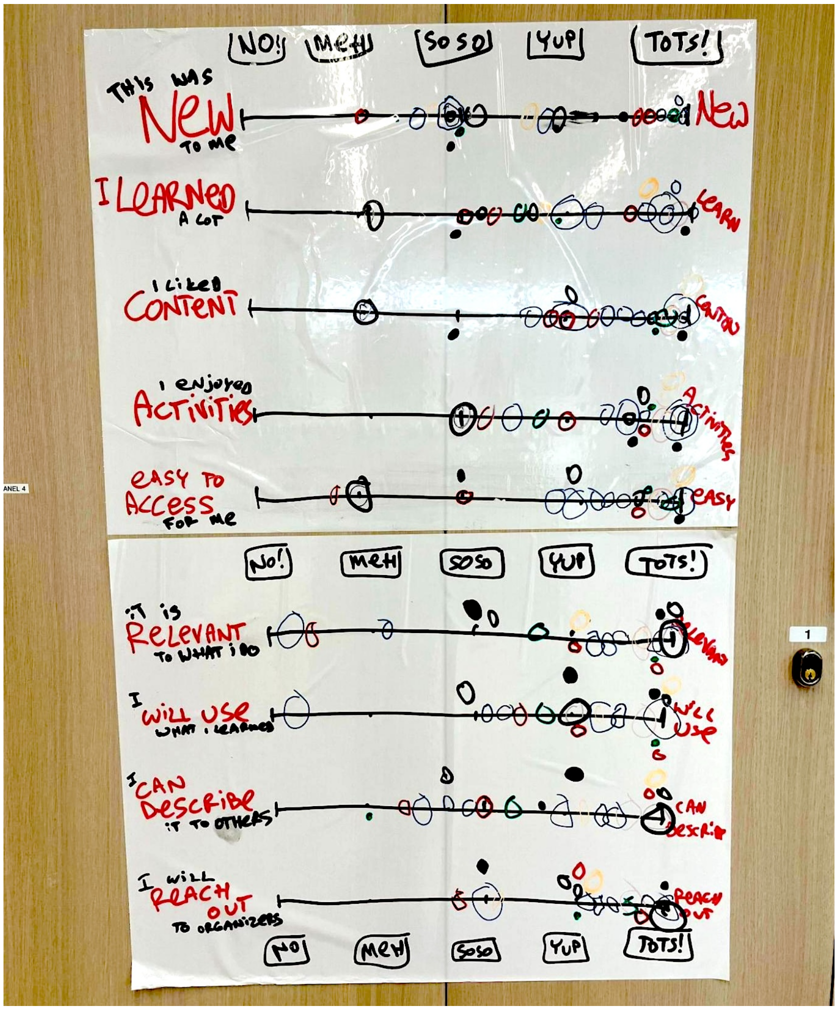

Figure 11.5: Feedback collected during a one-day intensive course on Hands-on Data Visualisation within Edinburgh Medical School at the University of Edinburgh.

that all student questions were repeated aloud during debrief sessions, while also encouraging students to share their own solutions via the chat.

At specific points in the course, we also asked all students to type a quick, micro contribution into the chat all at the same time, i.e. a Chat Blast (see Figure 11.6), to encourage engagement and shared participation. We took the Three Stars and A Wish approach, which is used by teachers in primary schools to help pupils focus on what is working and what needs improving (Larson et al. 2008). In our case, we flipped this around and asked students to provide us with "Three stars and a wish" about the course in the chat of the video call. This enabled them to share their thoughts not only on what they liked about the course but also gave them permission to point out things that could be improved or things they struggled with.

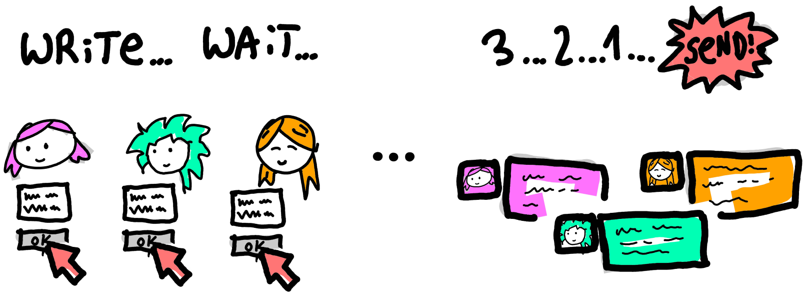

Figure 11.6: Chat blast with feedback using the "Three stars and a wish" method.

Using the Chat Blast method, we exposed the fact that we were learning from the students as well. We were constantly adjusting the course in an agile way according to the feedback received from students when possible. We also exposed the fact that not everything goes right all the time, that we are human beings who can make mistakes and are learning how to adapt to each new group of students in a fusion teaching environment.

11.4 A Practical Guide for Fusion Educators

After two intensive days in the trenches of fusion teaching, we emerged with battle-tested insights that we wish we had known before we started. Here are our concrete recommendations for educators considering teaching pair programming in a fusion (or hybrid) setting:

Before You Begin:

- **Rehearse every scenario.** Practice switching between online and on-site interactions until it becomes second nature. We literally role-played "How do I help a mixed-mode pair when one student is confused?"

- **Audio is everything.** Invest in quality headsets and microphones for all teaching staff and test for feedback loops before the course begins. Poor audio will sink your fusion dreams faster than any pedagogical mistake.

- **Designate an online advocate.** Assign one person the specific job of watching for raised hands, chat messages and overlooked online participants. Without this, online students become invisible.

During the Course:

- **Start mixed, then adapt.** Begin with mixed-mode pairs when energy is highest, but don't be afraid to switch to same-mode pairs if fatigue sets in. Flexibility is not failure; it is responsive teaching.

- **Feedback must be quick and visual.** Our slider system took 30 seconds to complete and showed results immediately. Long surveys kill momentum.

- **Normalise the weird moments.** When your voice appears "ghost-like" in someone's headphones, acknowledge it with humour. Students need permission to find fusion teaching delightfully strange.

Technical Essentials:

- **Font size matters.** What looks readable to you will be too small for online partners watching code.

- **Avoid unnecessary scrolling.** Every rapid scroll or sudden page jump becomes disorienting for online partners trying to follow along. Move deliberately through content.

- **Camera angles are crucial.** Online students want to see both your face and the classroom; position cameras accordingly.

- **Have a backup space ready.** Whether it is a smaller room, or yes, even a "broom cupboard," you'll need overflow space for mixed-mode pairs.

The hard truth is that some days, same-mode pairs will work better than mixed-mode ones. Accept this not as defeat, but as responsive teaching. The goal is not perfect fusion; it is creating a cohort that learns together, regardless of location.

Our most important lesson? Fusion teaching requires the same vulnerability we ask of our students. When we admitted our mistakes in real-time and adjusted together, that's when the real learning, for all of us, began.

11.5 Conclusions

Fusion teaching challenges traditional classroom dynamics, but with thoughtful design, it can create an engaging and collaborative learning experience. By experimenting with different interaction strategies, we found ways to bridge the physical and digital divide between online and in-person students, creating an interactive and integrated cohort. Our lessons learned and practical insights are timely and valuable for educators designing hybrid courses, particularly in the post-pandemic education landscape.

Fusion teaching challenges traditional classroom dynamics, but with thoughtful design, it can create an engaging and collaborative learning experience. By experimenting with different interaction strategies, we found ways to bridge the physical and digital divide between online and in-person students, enabling effective pair programming partnerships across both modalities. The mixed-mode pairing approach, while challenging to implement in typical classroom environments due to background noise and cognitive load issues, demonstrated the potential for bridging physical and digital collaboration in programming education.

When logistical constraints made mixed-mode pairing difficult to sustain, our experience still provided valuable insights into preparing students for the increasingly common reality of distributed software development teams. Our lessons learned and practical insights are timely and valuable for educators designing hybrid programming courses, particularly in the post-pandemic education landscape, where graduates from various disciplines increasingly need coding and remote collaboration skills in their professional work.

12 Changing civil engineering students' mindset toward programming

12.1 Introduction

It is high time that students from civil engineering (CivEng) get on with some serious programming. Civil engineering involves creating solutions to real-world problems (Wood 2012). As such, each solution tends to be unique and associated with the particular environmental and societal circumstances pertaining to the local community. It is no surprise that computing had early been adopted in CivEng, with programming classes added to the curriculum, as reported in a US survey (Hoffbeck et al. 2016). After all, CivEng students cannot just sit there and look at their university friends doing all kinds of interesting stuff with code or AI/machine learning. But what are the difficulties behind the scenes? And why do I focus here on such a narrow engineering discipline?

Civil engineers have good logical ability, especially deduction, using and manipulating mathematical expressions, dealing with real dimensions, approximation and optimisation. They have a firm belief in the deterministic: an expected system for this input must yield that output.

Until they start to realise that the computer system (or the compiler, but anyway the students don't care about this) tends to play tricks on them. For example, a code snippet copied from the online tutorial doesn't work, or merging two innocent-looking code chunks throws an error. That's when they realise that they will need a good approach to programming. The class instructor, too, will need to dedicate time to create good code examples as teaching materials — perhaps examples that illustrate some key principles in programming but merge seamlessly with domain knowledge.

This chapter presents some of my experience in tutoring two classes in Thuyloi University (Vietnam), one for final-year undergraduates and one for an MSc class, with use of the Python and Julia programming languages. In both cases, instead of delving into the merits of each language or making any comparison between them, I focus on the practical aspects of solving a typical civil engineering problem — sediment transport. For this problem, we are interested in the amount of sand moving in natural bodies of water (e.g. lakes, rivers) and how the bed elevation of such lakes/rivers changes over time, which has vast applications in real development projects. There is some maths and physics involved, but I will provide simple explanations. Hopefully, by the end of the chapter we can pinpoint some "good practices" for engineering students starting to write their own code. The chapter may also benefit CivEng class instructors who are aiming toward more effective lectures.

12.2 Code Implementation

12.2.1 Logic of calculation

CivEng students generally do not encounter any difficulty in implementing a series of formulae in order, as well as translating a formula to code. For example, in the Python snippet below, the drag coefficient is readily calculated according to the math equation:

<table>
<tr><th>Math</th><th>Code</th></tr>
<tr><td>$C_D = \left[\frac{0.4}{1+\ln(z_0/h)}\right]^2$</td><td><pre>CD = (0.4/(1 + math.log(zo/h)))**2</pre></td></tr>
</table>

CivEng students are well aware of complex engineering formulas and would be willing to write lengthy statements for mathematical expressions. Furthermore, they have adopted the style of naming variables according to the symbolic representation which is widely agreed in the literature. For example, here the name `CD` is used instead of choosing a more explicit name like `drag_coefficient`.

Even a branching construction would not cause much difficulty, for example to calculate the critical flow velocity using Python. The CivEng student surely would not get confused with units during computation, as they can handily type in conversion factors along with the code:

<table>
<tr><th>Math</th><th>Code</th></tr>
<tr><td>$U_{cr} = \begin{cases} 0.19\,(d_{50})^{0.1}\log_{10}\left(\frac{4h}{d_{90}}\right) & \text{for } 0.1 \le d_{50} \le 0.5\ \text{m} \\ 8.5\,(d_{50})^{0.6}\log_{10}\left(\frac{4h}{d_{90}}\right) & \text{for } 0.5 \le d_{50} \le 2\ \text{mm} \end{cases}$</td><td><pre>if 0.1E-3 <= d50 <= 0.5E-3: # mm
  Ucr = 0.19*(d50**0.1)*math.log10(4*h/d90)
elif 0.5E-3 < d50 <= 2E-3: # mm
  Ucr = 8.5*(d50**0.6)*math.log10(4*h/d90)
else:
  print('Warning: d50 out of range to apply formula</pre></td></tr>
</table>

12.2.2 Dimensions and units in civil engineering

A particular strength of the CivEng student is having keen eyes on physical quantities. The branching statement presented above shows that the student was aware of units and applied corresponding coefficients. The ability to tell meaning from numbers allows them to check intermediate results and correct potential errors during computation. Another advantage is that by adhering to the rules of dimensions in physics, students naturally familiarise themselves with the type system and therefore will not have much difficulty with languages like C#, Java, or Scala, or writing type annotations in their Python code.

12.2.3 Matrix manipulation

In many circumstances, the CivEng student has to deal with matrices, such as when solving partial differential equations (PDEs). One of such typical equations is the mass balance equation, where the (time) change in "stock" is related to the spatial changes (gradients) of "flows". It is not too difficult for the CivEng student to understand the finite difference technique and implement a naive version to solve PDEs. As an example, in the problem of sediment transport in rivers or seas, we consider solving the equation for z_b (the underwater bed elevation), which changes in time, given the two fluxes of sediment, represented as two components (q_x and q_y) in the two horizontal directions, x and y. It is helpful to visualise the river bed or seabed as a "tiled" surface where each cell has its own elevation z_b. See the diagram (Figure 12.1) for a simplified representation of the quantities involved.

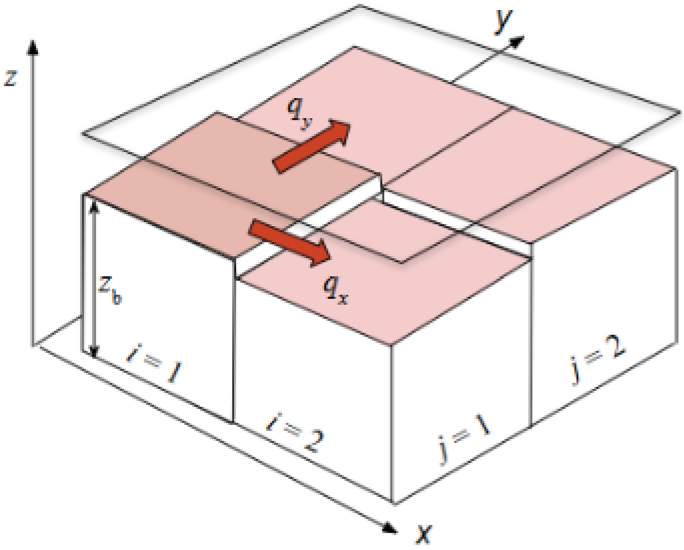

Figure 12.1: Sediment transport

It should be noted here that the CivEng student is equipped with numerical analysis skills and can approximate the differential in terms of finite differences.

Differential equation

$$\frac{\partial z_b}{\partial t} = -\frac{1}{1-n}\left(\frac{\partial q_x}{\partial x} + \frac{\partial q_y}{\partial y}\right)$$

where z_b, q_x, and q_y are functions of x and y. Therefore, we will also make $\frac{\partial z_b}{\partial t}$ a function of x and y.

Numerical approximation

$$\left(\frac{\partial q_x}{\partial x}\right) = \frac{q_{i,j} - q_{i-1,j}}{dx}$$

Here the numerical approximation for one flux gradient term is illustrated.

The naive approach is to use a nested list to emulate a 2-D array, which doesn't seem very efficient. But that's enough for the CivEng who wants to get the job done. This way, the code is more transparent when using an index system with $(i \pm 1)$. Notably, there is similarity in operators and operands between Python and Julia (thus I will place ellipses in the Julia code). The differences are in for-loop syntax and the 1-based indexing system in Julia as opposed to Python's 0-based. A notebook implementation[1] is provided online.

Python

```
for i in range(1,nx):
  for j in range(1,ny):
    dqxdx[i][j] = (qx[i][j]-qx[i-1][j])/
    dqydy[i][j] = (qy[i][j]-qy[i][j-1])/
    dzbdt[i][j] = -(1/(1-n)) * (dqxdx[i]
```

Julia

```
for i = 1:nx
  for j = 1:ny
    dqxdx[i,j] = ...
    dqydy[i,j] = ...
    dzbdt[i,j] = ...
  end
end
```

The use of matrix operations (e.g. using NumPy[2] arrays in Python) is faster compared to the "naive" version above, usually quoted as 10 times (Langtangen 2006). On the downside, the matrix notation obscures the indices `i` and `j` commonly found in numerical methods (see Figure 12.2, where part of a stencil[3] is shown. In this part, where arrows represent subtraction, we are finding "local" difference in the same `qx` array). When using matrix manipulation, the bounds must be specified correctly in all matrix subscripts (see Figure 12.3 and the code below). So, there should be a consideration in choosing either approach.

Looping through index

[1] https://nbviewer.org/url/coastal-study.uk/sed.trans/SedTrans.ipynb

[2] https://numpy.org/

[3] A stencil is a diagram showing a group of computational nodes where numerical approximation takes place.

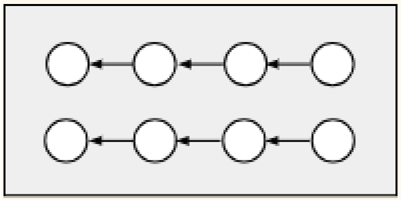

Figure 12.2: Lattice 1

Python	Julia/Python
`dqxdx[i][j] = (qx[i][j]-qx[i-1][j])/dx`	`dqxdx[i,j] = (qx[i,j]-qx[i-1,j])/dx`

Matrix-based computation

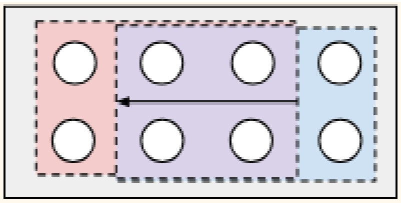

Figure 12.3: Lattice 2

Python	Julia
`dqxdx[0:-1,:] = (qx[1:,:] - qx[0:-1,:])/`	`dqxdx[1:nx-1,:] = (qx[2:nx,:]-qx[1:nx-1,:])/dx`

Another source of confusion is due to the concept of "views" in NumPy[4], for which the submatrix, e.g. `dqxdx[0:-1,:]` in the code snippet above, is a *view* and not quite a *copy* of the array `dqxdx`. In essence, NumPy "views" will behave in unexpected ways. Sure, this can be explained to the students, but every such issue can add up and discourage the learners. Therefore, an "agreement" might be made beforehand, that the matrix `qx` here is calculated once and cannot be modified thereafter. Otherwise, you must switch to another programming language, e.g. MATLAB, although this comes with its own issues.

12.3 Code reuse

To CivEng students, reusing code is actually more difficult than writing new code! While the students can implement the code quite well in terms of logic and calculation (see the previous section), they somehow feel embarrassed when having to perform the calculation again with a slightly different input data, or in the CivEng's language, another "scenario". Given the intuitive layout of code blocks in a notebook, the students tend to just copy-and-paste code and make minor edits, which leads to code with excessive repetition. Other related "gotchas" are discussed in other chapters of this book (Gemayel et al. 2026; Skipsey et al. 2026).

During my tutorials in the past, students were exposed to a limited amount of pre-written code in the form of Jupyter notebooks only. By reading these Jupyter notebooks, the students were

[4] https://numpy.org/doc/stable/user/basics.copies.html

able to pick out relevant code cells and run them. They would know which chunks of code should be executed and in what order, so that no errors emerge showing undefined variables. But when students copy chunks of code from the sample file to their *own* notebook, renaming variables is often required.

Another issue is lexical scoping. Being more acquainted to “physical” objects rather than information “pieces”, CivEng students would insist on using different names for variables appearing in both the inner and outer scopes despite the fact that these names would not conflict. This is reasonable for array data types, as in the case below, where the CivEng student creates a `zb_init` variable to store the initial state of the system (and the use of `copy()` to avoid the potential issues with using matrix views mentioned above). However, the situation is different for single numbers (scalars). Consider a variable for the characteristic “near-bed” flow velocity, `Uf`, which is created outside and being updated inside the function body, without affecting its original value.

```
zb = np.zeros((nx,ny))
zb_init = np.copy(zb)  # initial value
dt = 2.0  # const param
S = 1E-4  # const param
zs = 1.0  # const state variable
Uf = np.sqrt(9.81 * zs * S)


def evolve():
  for i in range(1,nx):
    for j in range(1,ny):
      zb[i][j] += dzbdt[i][j] * dt

  Uf = np.sqrt(9.81 * (zs - zb.mean()) * S)


for _ in range(nt):
  evolve()


print('Initial zb:', zb_init)
print('Initial Uf:', Uf)
```

12.4 Code development

12.4.1 Mental orientation

An important aspect of moving from a CivEng to a programmer is to re-imagine the conceptual model for the system. Compare:

	CivEng	CompSci
Paradigms:	imperative, procedural	object-oriented
Object type:	concrete, physically meaningful	abstract, blueprint to generate instances
Action:	math and logic operations applied to parameters of the object	objects' methods invoked on instances

Admittedly, it is not easy to persuade the student to adopt an OOP (Object Oriented Programming) style of programming to benefit future use. While for certain disciplines, e.g. Chemistry, the notion of class is inherent (e.g. class of elements), the CivEng is more inclined to think about real-world entities: a particular building, canal, etc. to be modelled.

Unfortunately, the common situation for CivEng is that many handbooks and guidelines adopt the procedural approach which can be summarised as follows:

1. Check the set of inputs against some predefined criteria.
2. Follow an established method step-by-step (such as the one in the SedTrans notebook[5] mentioned earlier) to arrive at some output (such as the vertical change in seabed level, `dzbdt`).
3. Evaluate the result, `dzbdt` — is it in a reasonable range? Is it realistic, and how does it compare to similar studies in the past ("literature")? Furthermore, is the computational routine *stable*, i.e. the numerical error does not grow up unbounded, through many iterations.

Apparently, most of the computational workload here lends itself to imperative programming. We could think of OOP aspects in organising a class of problems presented in step (1), or a class of test cases in step (3), but that is all. CivEng still relies on researchers in physics whose perspectives clearly align to the model: data in → transformation → data out.

[5] https://nbviewer.org/url/coastal-study.uk/sed.trans/SedTrans.ipynb

12.5 Generalisation

CivEng students understand the use of functions and are adept in translating formulation of various calculation routines into functions. So that should be enough for generalising code, shouldn't it? During a practice session, after having students estimate the derivative $f'(x_0)$ for the specific functions $f(x) = x^2$ and $f(x) = \cos(x)$ with finite differences, I required them to generalise their `deriv_*` function to accept any mathematical function. Well, how can we do that? — they doubted, until I showed that we can just pass the function under investigation as one of the inputs of the function `deriv` (see the Julia code below) — a first step toward functional programming.

Listing 12.1 Student's version

```
deriv_sq = function(o, h)
  ((o + h)^2 - (o - h)^2) / 2h
end

deriv_cos = function(o, h)
  (cos(o + h) - cos(o - h)) / 2h
end

deriv_sq(0, 1E-3)
deriv_cos(0, 1E-3)
```

Listing 12.2 Recommended version

```
# A generic central-difference approximation
deriv = function(f, o, h)
  (f(o + h) - f(o - h)) / 2h
end

deriv(x -> x^2, 0, 1E-3)
deriv(x -> cos(x), 0, 1E-3)
```

12.6 Tooling

The students are happy with Jupyter notebooks during class. Syntax highlighting, code auto-completion, and embedded plot outputs are features that are highly appreciated. Nevertheless, a dedicated environment like VS Code is needed for students who want to develop larger programs with debugging and version control (e.g. using git). In another chapter of this book, Gemayel et al. (2026) mention the use of a Python debugger (`pdb`) in the Jupyter Notebook environment, yet the more recent Jupyter Lab features a more convenient visual debugger so that students can track the changes of variables during the flow of the program.

Recently, there has been an increasing tendency to use AI in coding. While AI tools are a great boon for syntax checks and documentation, it would be detrimental to have AI chatbots generate the code skeleton and logic, as this part requires the ingenuity of civil engineers who possess the domain knowledge.

12.7 Good practices

This list is non-exhaustive yet would be useful for CivEng students starting to develop their code:

- Variable names should follow the widely adopted **notations** in civil engineering. When possible, use the Unicode characters for Greek letters, e.g., using instead of `alpha` (subject to the language's feature — this is favourable for Julia with keyboard shortcuts for such characters/symbols).
- Extensive and meaningful **documentation** is needed since there are many empirical formulae used in CivEng and the choice of a formula should be justified based on the context.
- Whenever the names are cluttered, consider using **namespaces** (knowledge of the engineering specialisation is required.)
- If facilitated by the language (in the case of Julia), prefer using **loops with index variables** (`[i±1]`) rather than implicit array position index (`[:]`).
- Generalise code with **functions**, so that you can reuse it later. This is particularly useful in CivEng, which heavily relies on various recipes. Each recipe can be implemented as a function.
- Using **advanced features** in Jupyter Lab or an IDE to organise and debug code (first step), and then aim toward collaboration, version control and testing.

12.8 Final thoughts

In summary, although there is not much material available for CivEng students' code development, there are patterns they may adopt to boost their performance at work. In this chapter, I present some such common patterns. Due to the varied nature of the disciplines that a CivEng may be involved in, it by no means presents a single context that they can build programs upon. Rather, I present a case of water/hydraulic engineering and sediment transport, showcasing field variables, spatial derivatives and implementing a finite difference numerical solution.

Given the ubiquity of materials and tools to learn and practice programming nowadays, CivEng students can leverage their skills to solve specific problems in their field. However, unlike data science where standardised software libraries have been developed, CivEng students may often find themselves coding from the ground up. In such situations, being able to design their program is crucial; and this can be accomplished through extensive practice with good programming patterns and continually learning to shift the mindset from “computing with numbers” to dealing with data organised on your computer. This is not easy, though, and I would refer to Mindstorms (Papert 2020b), where the author used Turtle’s geometry to dispel the “mathophobia” among schoolchildren.

These days, with programming a mandatory part of CivEng students’ skill set, it is best for them to appreciate programming as a way to represent their own workflow and reach a solution to their engineering problems, which can then be evaluated and improved.

13 Overcoming coding anxiety: reflections and strategies

Coding is increasingly acknowledged as a core component of data analysis across disciplines, yet it is often presented to students as a tool rather than a primary focus of study. For these students, it is likely that they have no prior exposure to coding. These initial interactions will define a long-term relationship with programming, and as a consequence, how they navigate an increasingly data-centric world. Understanding and addressing barriers to learning coding is essential to support learning, and here we consider one major barrier: coding anxiety.

Coding anxiety, or programming anxiety is defined as ‘a psychological state engendered when a student experiences or expects to lose self-esteem in confronting a computing situation’ (Connolly et al. 2009). It may be driven, in part, by other anxieties, such as computer anxiety (Chua et al. 1999; Meinhardt-Injac and Skowronek 2022) or statistics anxiety (Zeidner 1991), among others. While one may assume that “digital natives” (Bennett et al. 2008) experience a lower level of computer anxiety, this assumption may not naturally translate to coding anxiety (Nolan and Bergin 2016). Factors such as variable exposure to coding, an over-reliance on connected digital ecosystems during childhood (*How’s Life for Children in the Digital Age?* 2025), or disciplinary background, may influence levels of anxiety. For example, different levels of coding anxiety exist for mathematics and biomedical undergraduate students (Miller and Pyper 2024), with the former finding coding to be more enjoyable, and hence having lower anxiety. More broadly, student identity and background can also contribute to anxiety-linked barriers to learning, such as gender (Forrester et al. 2022) or learning in a non-native language (Kaur and Newell 2024).

Given the multifaceted nature of coding anxiety (PAN and Harun 2025), here we draw on our collective experience of teaching coding, provide perspectives on coding anxiety, and offer strategies to mitigate it. We focus specifically on students for whom coding is not their core academic focus, but rather a necessary skill set required when taking classes involving statistics, data analysis, experimental design, and modelling.

13.1 Recognising and Refocusing Anxiety

Anxiety sustains itself through a self-perpetuating cycle (Mkrtchian et al. 2017). It is believed to start with a trigger (e.g., a bad experience), which leads to future avoidance of associated triggers in an attempt to feel some short-term relief (Hofmann and Hay 2018). However, avoidance can lead to heightened anxiety in the long term (Mkrtchian et al. 2017). Since the brain learns that avoidance decreases anxiety, even if only temporarily, it may eventually prevent engagement with the trigger entirely. If the trigger is computer code, then coding becomes something to avoid, and hence coding anxiety is born. In contrast, confronting the trigger can build confidence and reduce anxiety over time (Kampmann et al. 2018; Craske et al. 2014).

We believe that recognising coding anxieties upfront is an important first step towards demystifying it in the classroom. Once students realise that this is an identified problem, common to many rather than something affecting them alone, they are one step closer to overcoming it. Humour, including self-deprecating jokes, describing how, despite now being the teacher, you once suffered from coding anxiety, can go a long way towards making students feel better[1]. Cartoons may also be an effective way of making anxiety itself less daunting. If existing online content is not enough, and making cartoons is not one's gift, AI-generated images could easily be created for that purpose - just make sure people have 5 fingers!

Figure 13.1: Figure 1. An example of a humorous cartoon that normalises coding anxiety. (https://www.thesquarecomics.com/) TODO: attribute this properly

Given anxiety's negative feedback loop, the best way to combat anxiety is to break this vicious cycle (Figure 2). As an instructor or teacher, this can be achieved by providing positive experiences with coding[2]. Importantly, after acknowledging that coding anxiety is real, the

[1] Even if you've never experienced coding anxiety - lucky you; most of us have, at least to some extent - do take one for the team, and say you have too.

[2] For some of us, ease with coding developed through repeated practice and the satisfaction of automating otherwise repetitive tasks. For others, it emerged more gradually and holistically, as their overall perception of coding improved over time through continued engagement.

solutions to it should be presented along with how overcoming their anxiety will be supported, e.g., reflecting on personal experience. Offering your experiences and providing a very visible role could serve as a powerful catalyst for students to realise they can overcome their own coding anxiety too.

Figure 13.2: 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/ TODO: TAM creating a new image to replace this one; will look about the same

Gauging a class's perception towards coding early should prove useful in lowering overall anxiety. Collecting information about *a priori* fears or causes of anxiety, including - but not exclusively - coding anxiety, as well as discussions about course expectations, provides clues to what might be good strategies to adopt. In-class feedback tools like Mentimenter[3], allow students to provide feedback anonymously and in real time to make the course more accessible in an informal, friendly, and interactive way. This helps generate useful ideas, including specific suggestions, for how to reduce anxiety. Additionally, having students reflect on how anxious they are about coding might allow them to identify shared problems and anxieties across the class; a problem shared is a problem halved.

13.2 Building Confidence Through Active Learning

Early hands-on activities focusing on the end product of coding, rather than the process of getting there has the benefit of demonstrating value without the need to actually code in the

[3] http://www.mentimenter.com

first instance. Even exercises where students simply copy-paste code, without knowing why it does what it does (but being told that later they will be able to code it themselves), could go a long way in reducing anxiety. Such exercises allow students to see how useful coding is for tasks they would like to do, but which would be impossible manually. Focusing on the outcomes students are interested in and can relate to makes them less reluctant to learn programming than if they just see it as a barrier. For example, one could take a couple of large datasets linked by a common variable and ask students to examine the files in a text editor and "understand" the data. Then the students can be given code to produce a captivating visualisation (e.g., Figure 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.

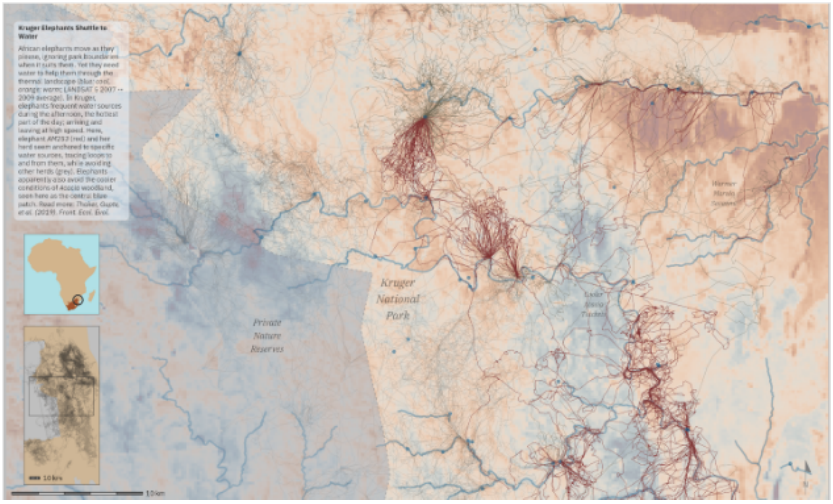
Kruger Elephants Shuttle to Water
African elephants move as they please, ignoring park boundaries when it suits them. Yet they need water to help them through the thermal landscape (blue: cool, orange: warm; LANDSAT 5 2007 -- 2009 average). In Kruger, elephants frequent water sources during the afternoon, the hottest part of the day; arriving and leaving at high speed. Here, elephant AM232 (red) and her herd seem anchored to specific water sources, tracing loops to and from them, while avoiding other herds (grey). Elephants apparently also avoid the cooler conditions of Acacia woodland, seen here as the central blue patch. Read more: Thaker, Gupte, et al. (2019). Front. Ecol. Evol.
Kruger National Park
Private Nature Reserves
10 km
10 km

Figure 13.3: Figure 3. Example of a beautiful visualisation created using R (a map of elephant movements through Kruger National Park). Produced by Dr Pratik Gupte for the British Ecological Society Movement Ecology Special Interest Group's MoveMap competition (2021), and reused here with his permission.

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, the package praise can also come in handy. A call to its single function `praise()` returns positive feedback like "You are wonderful!" or "You are supreme!".

These simple statements often put a smile on students' faces, and based on our experience, that is halfway towards reducing anxiety. Finally, if students just can't code but want to "fake it till they make it", the R function `look_busy()` in the `player` package will produce output on the console that will make anyone else believe students are working hard (showcase it, but discourage its use!). We have used several of these approaches and provide them as a starting point, but given R is a dynamic and ever-evolving environment, other fun and engaging options must live out there waiting to be used by teachers to help reduce code anxiety.

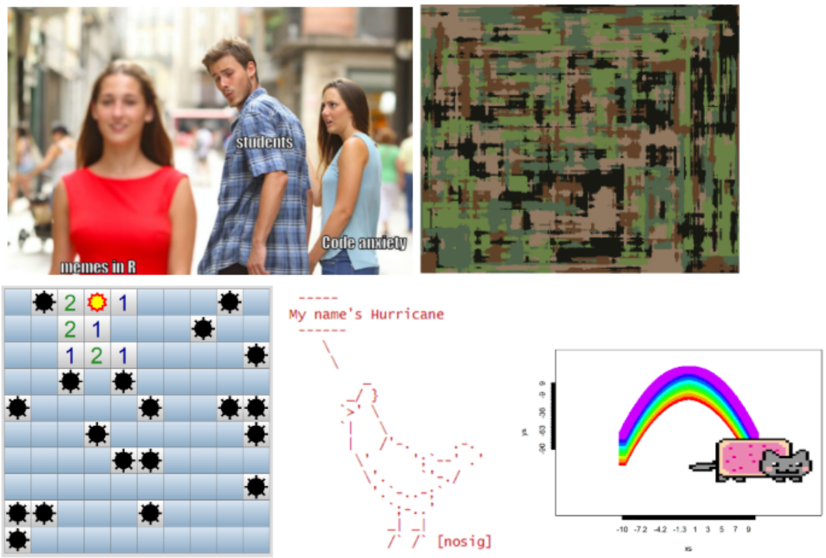

Figure 13.4: Figure 4. Examples of how R coding can be fun. A: A meme showcasing how memes can reduce code anxiety; this meme can be recreated with 2 lines of code using package **memer**[4]. B: illustrating P-values via emogi with package **emo**[5]. C: A painting created in **aRtsy**[6], a generative art R package. D: An instance of a minesweeper game created in package **fun**[7], demonstrating the authors inability to play it. E: An example of an animal that can be drawn in R syntax using the **cowsay**[8] package. F: The **CatterPlots**[9] package can be used to generate scatterplots that include cats.

In our opinion, live coding is one of the most effective tools to demystify programming and reduce coding anxiety. When instructors write, run, *and debug*, code in real time, narrating their thought process as they go (and getting error messages), they model both the practice and the mindset of coding. Students witness that even experienced coders make mistakes, encounter errors, and debug them systematically. This transparency transforms coding from a mysterious, high-stakes activity into a visible process of exploration, reasoning, and iteration.

Importantly, the approach taken during live coding matters as much as the content. Speaking aloud while typing, e.g., explaining why particular commands are chosen, predicting outputs before running them, and pausing to invite student predictions, turns what might otherwise be a passive demonstration into an interactive learning experience. When an error appears, embrace it as an opportunity to model problem-solving, showing that mistakes are normal, and even useful. Seeing how to respond to an error message with curiosity rather than panic can profoundly reframe students' emotional response to failure.

To lower anxiety further, live coding should begin with simple, visually rewarding tasks. Plot a small dataset, calculate a mean, or clean a short table. This way, students can follow the logic without feeling overwhelmed by syntax. Gradually, the exercises can build in complexity, moving from replication to modification and finally to creative application. When possible, students can be invited to suggest the next line of code, predict what will happen if a parameter is changed, or even take over the keyboard for short stretches.

Finally, recording or providing annotated versions of live coding sessions allows students to revisit material at their own pace. This ensures that the spontaneity of live teaching is complemented by the reassurance of reviewable, structured notes. By combining openness, interaction, and a healthy acceptance of mistakes, live coding transforms what might otherwise be a source of anxiety into a collective, confidence-building experience.

There can be multiple approaches to teaching code and statistical data analysis, and some of these might improve or contribute to coding anxiety (Colquhoun, Marques, et al. 2026). Conventional wisdom suggests that the first step is the hardest for any journey. In our experience, starting with small, manageable, and clearly described tasks, will make the initial experience with coding positive, which will in turn decrease anxiety levels when facing harder tasks. This is consistent with the strategy suggested by (Auker and Barthelmess 2020a). Later classes or assignments may contain less detailed guidance or be more open-ended, allowing students to explore their new found confidence in their coding abilities at their own pace.

13.3 Support and Collaboration Beyond the Classroom

Coding confidence grows most effectively outside of formal teaching time. Low-pressure practice, whether collaborative or individual, helps students reinforce skills and approach coding with curiosity rather than fear. This can be supported by structuring opportunities for independent study and peer interactions that sustain progress between classes. One simple but effective technique is to encourage students to maintain a short 'coding diary', allowing for normalisation of occasional frustration and celebration of key achievements.

The fear of revealing a lack of coding ability can be a powerful barrier, discouraging students from even attempting new exercises and progressing in their learning. Independent work outside the classroom, supported by carefully developed additional materials, can help overcome this. Providing a list of such resources will empower students to learn on their own. In our experience, a particularly interesting resource comes via the **R** package **swirl**[10]. Swirl "teaches you R

[10] https://swirlstats.com/

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' (#TODO ***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 (TODO ***link to other chapters***). Pair-programming is a useful strategy to lower coding anxiety levels (Fan et al. 2025). Nonetheless, these are not universal findings; for example, (Krizsan and Lambic 2024) found that while performance improved, it did not reduce anxiety levels. Online pair-programming might be less intimidating for some, although it is not clear whether the benefits associated with live pair-programming apply (Hafeez et al. 2023).

AI-assisted pair-programming, where a coder is paired with an AI agent, has become an accessible alternative to human-human pair-programming (Fan et al. 2025). This might be easier for students who would otherwise be reluctant to expose their difficulties to a colleague. While AI-assisted pair-programming has been shown to enhance motivation, reduce anxiety, and even improve performance, it does not fully match the collaborative depth and social presence achieved through human-human pairing (Fan et al. 2025). Regardless of the pair-programming mode considered, we believe that if well thought out and prepared, pair-programming activities can be used to lower coding anxiety levels in a range of students.

Figure 13.5: Figure 5. Pair programming and the analogy of a driver and a co-pilot #TODO (will be replaced with reworked image by TAM)

Students can be encouraged to further their learning journey by 'teaming up' with peers to continue to pair-programme outside of the classroom environment. This can help alleviate coding anxiety both through additional assistance in a familiar mode and also through the mechanism of shared endurance and a head-on approach to the anxiety barrier.

Ultimately, support beyond the classroom requires fostering autonomy and connection in equal measure. When students know they can access help, whether from peers, teachers, lecturers, or AI tools, they are less likely to interpret difficulties as personal failings. Instead, they begin to view coding as a shared and iterative process that need not come with coding anxiety but natural failure and success routines.

14 Key Takeaways

Coding anxiety is real, especially in students for whom coding isn't a core part of their disciplinary focus. This barrier should be recognised, and we have both a responsibility and an opportunity to normalise it in teaching settings and explain why and how the anxiety cycle can be broken. Overcoming coding anxiety should be presented as a personal endeavour, but one that instructors and peers, along with a growing number of resources, can support. For a teacher, effective methods for breaking the anxiety cycle are as pedagogically important as content creation. Instructors can become "anxiety diffusers", starting by exploring ways to provide rewards from coding activities. In particular, activities that allow students to see the potential and value of coding to achieve tasks they are inherently interested in should be considered. Providing encouragement and additional resources for learning how to code, and promoting additional high-rewarding coding activities, are likely to help further. Strategies to share the burden, like pair-programming, could further boost self-confidence in coding. Overall, different audiences and topics will require slightly different approaches, so engaging with students and intervening using the strategies discussed above will be fundamental for reducing coding anxiety and improving learning.

15 Funding

TAMs work is partially funded by national funds through FCT – Fundação para a Ciência e a Tecnologia, I.P., under CEAUL Research Unit, UID/00006/2025, DOI: https://doi.org/10.54499/UID/00006/2025

16 References

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

17.1 Introduction

TODO: keep this as introduction, or ditch it?

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

17.2 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.

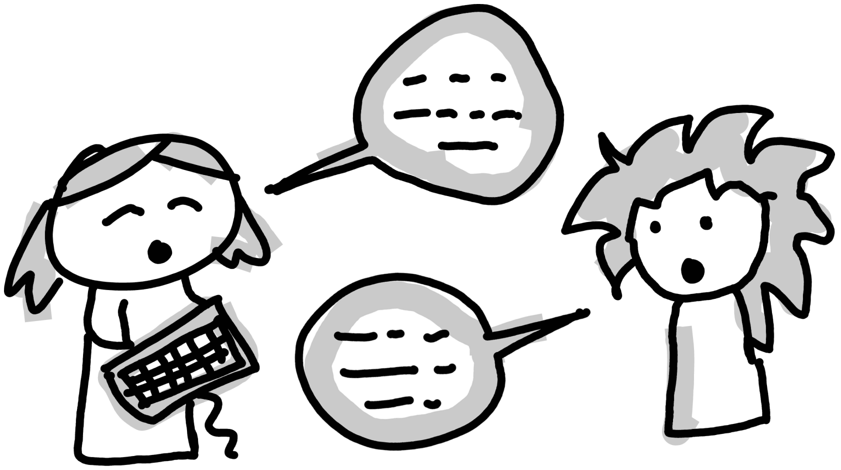

Figure 17.1: “I think your idea will work, let me type it in!”,” I see a bracket missing on line 7”, “Let’s run this code and see what happens.”, “Yay, we did it! It works!”

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.

17.3 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

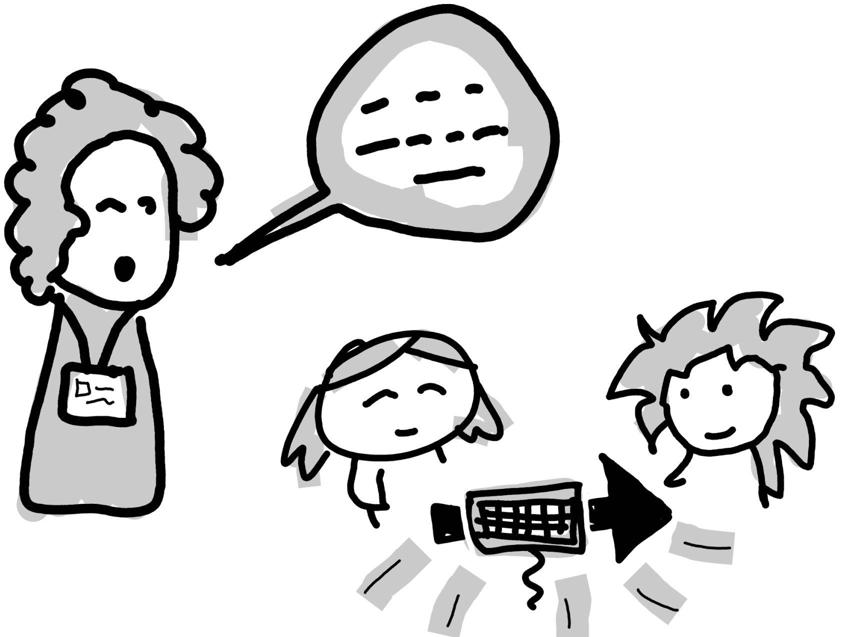

Figure 17.2: “Hello! Which of you is driving now?”, “Is it time to switch drivers?”, “Do you need help getting unblocked or are you just cracking on?”

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.

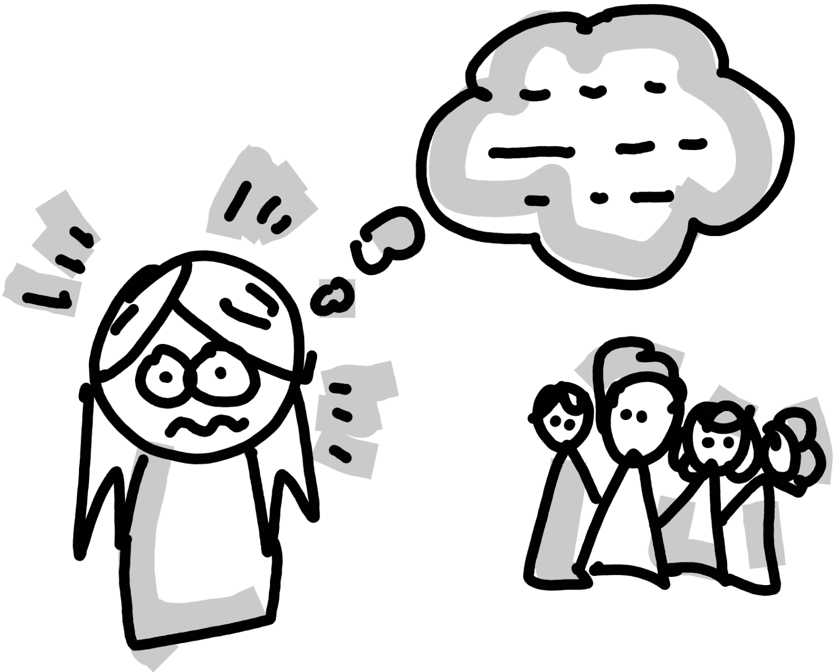

Figure 17.3: “What if I end up in a pair with someone who is mean, or much more advanced than me?”, “I’m uncomfortable in socially complex situations. Will I know what to say and do in a group…”, “Should I just always work with the same friend, or would I miss out on meeting new people, and learning 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.

17.4 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

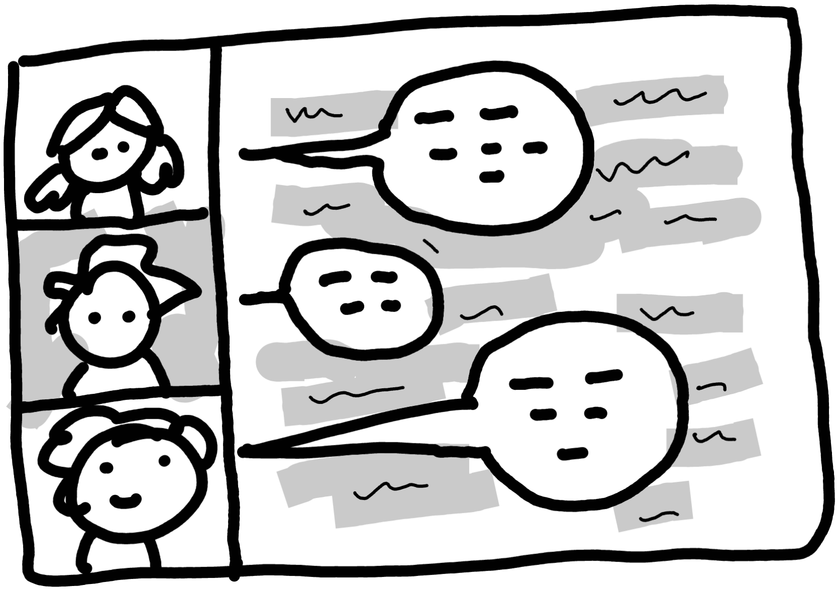

Figure 17.4: “So the way I understand it, is …”, “I see another way to solve this!”, “That’s cool! I never thought about it this way”

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

17.5 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.

17.6 This chapter was pair-programmed

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

18 The case for skills-based assessment in introductory programming courses

18.1 In this chapter

To learn to program is to develop a new set of skills. In this chapter, I'll discuss an approach to course design and assessment that brings skill development to the fore: the skills-based assessment framework. I'll explore what makes this framework particularly good for teaching programming to new programmers, especially ones facing cultural, structural, and institutional disadvantages. And I'll share a case study from my own context: I work as a Lecturer in Psychology at the University of Edinburgh, teaching undergraduate Psychology students how to do statistics using R. With my colleagues, I have developed an introductory course for these students using the skills-based framework, and our experience will be the focus of the final part of this chapter.

18.2 What is skills-based assessment?

A course built for skills-based assessment comes with an inventory of skills that the course developer wants their students to learn. For example, in our intro statistics course, skills include "I can represent binary predictors using treatment coding" and "I can transform a variable into z-scores"; in a more coding-focused course, skills might be something like "I can explain the difference between while loops and for loops" or "I can use pseudocode to write basic sorting algorithms".

The basic idea behind one possible variant of skills-based assessment, the one my team and I are using, goes as follows. As the course progresses, students work on developing each skill. They get multiple opportunities to show that they've gained each one. The only firm deadline is the very end of the course; until then, students can try to demonstrate a skill as many times as they need to or want to, and they won't be penalised at all for making multiple attempts. They get feedback on every attempt—why it succeeded or where it fell short. Once a student has successfully demonstrated a skill, then in the course's record-keeping, they have earned that skill permanently and cannot un-earn it. Each student's course outcome (whether number grade, letter grade, pass/fail, or something else) is some function of how many skills they earned.

This basic idea is known in different circles under different names. For example, Clark and Talbert (2023) and Buckmiller et al. (2017) call it “standards-based grading”; O’Leary and Stockwell (2021), (2022), and Zuraw et al. (2019) call it “skills-based grading”; and it’s also similar to a cousin framework called “specifications-based grading” (Nilson 2014; Clark and Talbert 2023).

I like the “skills-based” part of the name because it centres the skills, but I steer away from the “grading” part for a couple reasons. First, the skills-based course that my colleagues and I are developing will culminate in a pass/fail outcome, not a number/letter grade. I don’t want the term “skills-based grading” to mislead students about how they’ll be assessed. Second, I consider grades on the whole to be an unnecessary evil (see, e.g., Kohn 1993; Stommel 2023; Blum 2020), but “assessment” can be about assessing how a student is doing, about checking in and giving feedback on their learning journey—an approach which better encourages genuine learning (see, e.g., Ambrose et al. 2010; Sambell et al. 2012).

18.3 Why is the skills-based framework well-suited for teaching new programmers?

In this section, I’ll highlight three features of skills-based assessment that, in my view, make it much better than traditional weighted-averages grading for teaching programming to new programmers. Skills-based assessment:

1. reduces students’ academic anxiety and stress;
2. lets students learn at their own pace; and
3. helps students facing structural disadvantages to succeed.

18.3.1 1. Reduces students’ academic anxiety and stress

Many students, especially those from non-STEM disciplines, are anxious about learning to program (an observation to which a whole chapter in this volume is dedicated: Oldnall et al. 2026). And learning to program in a traditionally-graded course is like adding salt to the wound. Learning is messy, and mistakes are normal—arguably essential—along the way. But in traditionally-graded courses, mistakes are punished by lowering marks.

On the other hand, the skills-based framework acknowledges and encourages the messy process of learning. For example, allowing students the chance to demonstrate a skill repeatedly, with no penalty for re-attempts, fosters in them a sense that it’s okay to make a mistake and worthwhile to try again (O’Leary and Stockwell 2021). And students also gain a sense of self-efficacy as they realise they can accomplish something that they couldn’t do before (Zuraw et al. 2019). This leniency is a major reason why skills-based assessment has repeatedly been shown to lessen students’ academic stress (e.g., O’Leary and Stockwell 2021; Buckmiller et al. 2017; Zuraw et al. 2019; Lewis 2022).

In an ongoing research study, my colleagues and I are gathering data about how our skills-based course redesign impacts how our students feel about learning statistics and R programming. Based on the literature mentioned here, we hypothesise (and very much hope!) that our redesign will help ease our students' anxiety and stress.

18.3.2 2. Lets students learn at their own pace

Programming can be hard to learn. And while some people will take to it straight away, others need more time to develop this new way of thinking (Caspersen and Bennedsen 2007; Offutt et al. 2017). But students in traditionally-graded courses don't always have the time they need. Students typically lose marks if they haven't grasped a concept by the time the course developer thinks they should have—even if they do figure it out later.

In skills-based courses, learners are not held to the course developer's timeline. Students can demonstrate skills at any point during the course, so learners who benefit from a bit more time to learn can still succeed. Offutt et al. (2017) report that letting students pace their own learning in an introductory computer science course has led to great results: students learn better and are reportedly disinclined toward academic dishonesty, because the time pressure that had motivated them to cheat has been lifted.

18.3.3 3. Helps students facing structural disadvantages to succeed

The culturally ideal programmer is an able-bodied white man. Anyone who diverges from this ideal, along whichever dimension or dimensions, has extra cultural and emotional work to do when learning to code.

For example, Carter and Jenkins (1999) observed a "growing perception that female students are weaker at programming than their male counterparts" (p. 3), noting that female students tend to be less confident than male students and to underestimate their own ability more. In my earlier experience as a young female programmer, and now as a female teacher with many young female students, not much has changed about these views in the nearly three decades since.

And adding in the intersectional experience of race, Rea (2022) recounts the striking words of Olivia, who organises coding bootcamps for marginalised and minoritised people:

> I'm teaching women how to code. But it's bigger than that. What I'm really teaching them to do is how to unlearn what they've been taught. A lot of women of color, or women in general, have not been taught that science, math, coding, you name it, is for them. You're unlearning a thing, and you're learning a new skill.

Further dimensions compound the challenge of belonging in traditional academic spaces, no matter what's being taught: physical disability, mental disability, care responsibilities, work duties, a different first language than the language of education. All of these disadvantages can present a serious barrier in traditionally graded courses, in which students are commonly graded on things that go beyond the course's content-related learning objectives, such as regular attendance, active verbal in-class participation, and (as mentioned above) the timeliness of their understanding.

But in the skills-based framework, barriers like these are softened. Where clear expectations and goals are provided, less background knowledge is required, making the space more inclusive to people with different backgrounds (Parker 2018).[1] As such, skills-based courses have been shown to achieve more equitable outcomes for a wider range of learners (e.g., by O'Leary and Stockwell 2021, 2022). I really hope that the more flexible structure of skills-based assessment will help our intersectionally disadvantaged students build their skills as well as their sense of self-efficacy in a more compassionate space.

Finally, a note for even the culturally ideal student of programming: any person at any time could experience sickness, injury, or grief, or may just need time away to, say, bring their cat to the vet. Everybody benefits from inclusive pedagogy.

For further discussion of inclusion and accessibility in teaching programming, see Colquhoun, Ahern, et al. (2026) in this volume.

18.4 Case study: Developing a skills-based course introducing statistics using R

For all the reasons discussed above and more, my team and I in Psychology at the University of Edinburgh are using the skills-based framework to redesign our undergraduate statistics curriculum. The redesigned first-year course (link to the University of Edinburgh course catalogue DRPS)[2] will launch in September 2026, so as of this writing, no students have experienced it yet. In this section, I'll describe the path we took (largely guided by Clark and Talbert 2023, chap. 11) and some key decisions we made along the way.

[1] A well-formulated rubric may accomplish a similar goal, but rubrics tend to have two issues that skills do not. For one, by their nature, rubrics focus on the work's deficits until the very highest levels of achievement. Here is an example from one of our earlier report-based assessments: "Results are mostly clear and logically presented. Interpretation is mostly accurate but may lack depth in linkage to analysis strategy". A rubric targets all the ways the student's product falls short from the ideal. And for a student who already feels like they don't belong, deficit-focused approaches like this will not make them feel more welcome. And for another, imagine being a student and receiving the feedback: "to get a higher mark, you need to add more depth in the linkage to analysis strategy". What does it *mean* to "add depth"? And how will you know if you've successfully done it? Rubrics tend not to be as explicitly actionable as skills, and they tend to require background academic knowledge to successfully interpret—knowledge which educators take for granted but which not all students will have.

[2] https://www.drps.ed.ac.uk/26-27/dpt/cxpsyl08013.htm

18.4.1 Our development process

We began by dreaming up, in very broad strokes, what topics and methods we wanted to cover in our ideal version of the course. Our early notes included, for example, “intro tidyverse”, “descriptive stats”, and “simple linear model”. Once we all agreed on a set of topics and a rough order in which they’d appear, we dove into the weeds to develop a finer-grained list of things we want students to be able to do, such as:

- Install packages in R.
- Create new columns in a dataframe.
- Summarise numeric variables using the mean.

We ended up with a list of skills numbering between 80 and 100. This list grew and shrunk and changed over time as our ideas kept flowing. These are far from the set of skills we’ll actually assess, but this early representative list of skills was absolutely good enough for the steps that would follow.

At the same time, we also considered the course’s submission and feedback systems. How and when can students evidence skills? What kind of feedback will they receive?

We observed that many skills could be automatically checked (for example, identifying the code that would accomplish a given task), while a handful would require human review (for example, writing up a description of a given data set).

We decided to check most skills automatically using quizzes on our learning management system Blackboard Learn. These quizzes pull from a pre-built question pool for each skill. If students answer a question correctly, then that counts as evidencing the given skill. After submitting their answer, students will see automated feedback about how to solve the question and common misconceptions.

We decided to check the remaining skills that required human review at two points, one at the end of each semester. Students would be able to submit their responses (also on Learn) at any time, but to keep marking overhead reasonable, we only review their responses and give feedback twice a year. The feedback will be uniform across all students and simple, something like “successful”, “almost”, or “not yet” (Clark and Talbert 2023). For more specific guidance, students will be able to talk to us during the weekly coding workshops and our office hours.

Once these basic structures were in place, we began properly consulting with stakeholders (Sackstein 2015). From the beginning, we had support from the head of our subject area, which made every other consultation much simpler. We asked for input from the following groups of people:

- Students (but don’t expect a huge turn-out—we hosted an in-person drop-in feedback session, to which two of our approximately 400 students came, and we also sent out an online feedback form, to which four students responded).

- Academic staff (we hosted a staff-focused drop-in session and sent out an online feedback form, which altogether yielded a handful more responses from staff than from students).
- Professional services staff (especially the very knowledgeable people who deal with the back-end of inputting student grades into the institution's software system; they helped us identify and solve problems we had never considered).

After integrating the very useful input we got from all these people (examples of which I'll mention below), we submitted the proposal to the committee that processes course changes.

At that point, with the administrative overhead off our desks for a while, we started to fine-tune the skills that the first iteration of the course will actually assess.

18.4.2 Top tips for writing assessment skills

Designing the skills themselves is a creative logistical challenge with many degrees of freedom. In this section, I'll walk through how we resolved two major questions that came up repeatedly along the way: how specific vs. how high-level should each skill be? And what level of cognitive engagement should each skill call for?

18.4.2.1 Granularity of assessment skills

In our early list of topics, we wrote, for example, that we want students to learn to summarise data using the mean, the median, and the mode. How many skills should we use to assess this learning?

One option is to use one skill per measure:

- I can summarise numeric variables using the mean.
- I can summarise numeric variables using the median.
- I can summarise categorical variables using the mode.

Another option is to consolidate both numeric summary measures into a single skill:

- I can summarise numeric variables using appropriate measure(s) of central tendency.
- I can summarise categorical variables using appropriate measure(s) of central tendency.

A third option is to consolidate again:

- I can summarise variables using appropriate measure(s) of central tendency.

In principle, one could consolidate arbitrarily far, depending on the focus of the course: from "I can conduct an exploratory data analysis" up to "I can analyse data", or even beyond. Here, for the purposes of illustration, I will focus on the lowest three levels, since they are the most pertinent ones for our specific aims.

The relationship between these different levels of skills is shown in Figure 1, along with some useful vocabulary offered by Clark and Talbert (2023, 159).

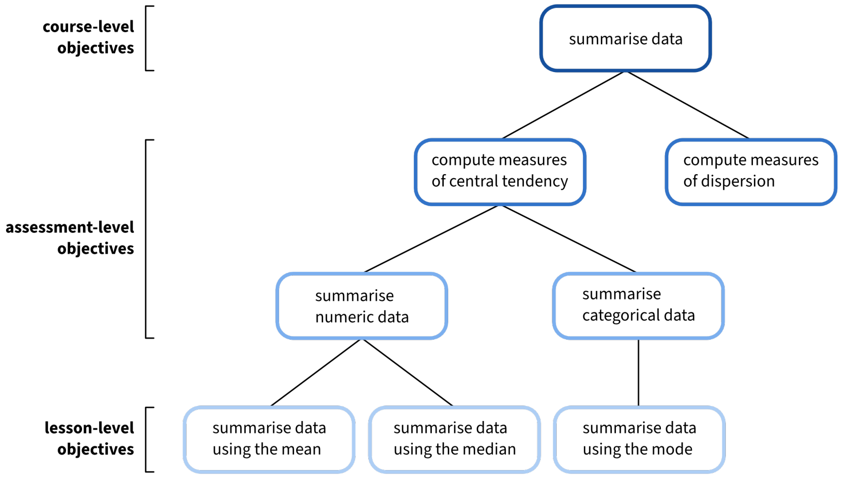
course-level objectives
assessment-level objectives
lesson-level objectives
summarise data
compute measures of central tendency
compute measures of dispersion
summarise numeric data
summarise categorical data
summarise data using the mean
summarise data using the median
summarise data using the mode

Figure 18.1: Figure 1. An example hierarchy of data summary skills, annotated with the levels described by Clark and Talbert (2023): course-level objectives, assessment-level objectives (a subset of which are the skills we want to assess), and lesson-level objectives.

At the very top of this hierarchy, in darkest blue, are what Clark and Talbert call the "course-level objectives": the broad, encompassing learning objectives for the entire course. Our running example belongs to a course-level objective like "I can summarise data", an objective which involves measures not just of central tendency but also of dispersion, for example.

Jumping to the very bottom of this hierarchy, in lightest blue, we see the "lesson-level objectives": what we want students to learn from individual lessons. Lesson-level objectives map fairly cleanly to the fine-grained list of learning objectives that we drafted early on.

In between these two extremes, in medium blues, we see the "assessment-level objectives". Somewhere in this middle ground are the skills that we will assess.

Do we want to make absolutely sure that students can summarise numeric and categorical variables individually? If we do, then a larger quantity of lower-level skills may be the way to go. Or do we assume that if a student can correctly find a sample's mean, say, they can probably also find the median and mode? If that assumption seems reasonable, then the higher-level skill might suffice.

In a coding course, an analogue to this quandary might be the following: Will students be assessed on appropriate use of control structures, broadly speaking? Or separately on being able to appropriately use if/then/else statements and various kinds of loops? Or, finer-grained, on specific structures like if/then, if/else, for loops, while loops, and so on?

In our case, for the data summary example, it is reasonable enough to assume that a student who can find the mean can probably also find the median and the mode. And further, we are also trying to keep the sheer number of skills in check (the current list, as of this writing, contains about 40). We therefore chose to assess the most consolidated, broadest, highest-level version of skills wherever we could.

18.4.2.2 Cognitive level of assessment skills

Each skill should call for an action that a student can easily understand and that an assessor can easily observe (Clark and Talbert 2023). But for a given skill, we found that often, several different actions will all seem reasonable. For example, after students learn about t-tests, do we want them to be able to *conduct* a t-test, or to *explain* the logic behind a t-test?

Choosing the action verb is a weighty decision. Different verbs, like "conduct" vs. "explain", invoke different levels of cognitive engagement, as we know from frameworks like Bloom's taxonomy (Bloom et al. 1956; Anderson and Krathwohl 2001) and the SOLO (Structure of the Observed Learning Outcome) taxonomy (Biggs and Collis 1982).

In an early version of the course design, we had imagined that students could achieve one of three possible course outcomes: Merit, Pass, or Fail. This design allowed us two sets of skills: a "core" set of skills, which would cover simply conducting statistical tests, as well as an "advanced" set of skills, which would add on the deeper conceptual understanding. If students demonstrated both core and advanced skills, they would receive a "Merit" outcome; if they demonstrated only core skills, they would receive a "Pass". (I note that pedagogically, dangling the "Merit" carrot may have many of the same demotivating effects as traditional grading—see, e.g., Kohn 1993.) We discovered, though, that a categorical three-level course outcome is impossible within the strictures of our institution's software system—one example of many to illustrate why consulting with professional services colleagues is absolutely key.

In the later version of the course, we narrowed the outcomes down to Pass and Fail (an option welcomed by several students we talked to, who liked that it frees them from the stress of "chasing an A"). With no "Merit" category, we set aside the core vs. advanced sets of skills. Again, we faced a choice between "conduct" and "explain"—but again, Clark and Talbert (2023) offer guidance. They comment (p. 165) that hands-on, skills-building courses tend to use vocabulary from the more fundamental, less abstract levels of Bloom's taxonomy, whereas more conceptually-focused courses tend to use the more abstract, higher-level verbs. We want our students to focus on building skills, so we opted for the more hands-on "conduct a t-test" variant throughout.

18.5 The big skills-based picture

To conclude, let's zoom back out: what makes skills-based assessment well-suited for helping students learn to program? The picture I hope to have painted in this chapter shows a different way of checking in on how people learn, one that genuinely recognises students as the learners they are, not just as faulty repositories for the information we bestow (Freire 1996). Because skills-based assessment mitigates common causes of anxiety and softens many structural barriers that students face, it is our framework of choice for helping new programmers start their learning journey.

18.6 Acknowledgements

This chapter would not exist without:

- Franziska McManus, who first suggested I share these ideas in this book.
- Itamar Kastner, who introduced me to skills-based assessment and who continues to inspire the best parts of my practice.
- Emma Waterston, Josiah King, and Umberto Noè, my colleagues on the Psychology stats team, who are all incredibly caring educators and fun, thoughtful, and thought-provoking collaborators.

Bibliography

Aho, Alfred V. 2012. “Computation and Computational Thinking.” *The Computer Journal* 55 (7): 832–35.

Alex, Beatrice, Clare Llewellyn, and Pawel Orzechowski. 2026. *Teaching Programming Across Disciplines.*

Alex, B., C. Llewellyn, P. M. Orzechowski, and M. Boutchkova. 2021. “The Online Pivot: Lessons Learned from Teaching a Text and Data Mining Course in Lockdown, Enhancing Online Teaching with Pair Programming and Digital Badges.” *NAACL-HLT 2021*, 138.

Allison, Truett, and Domenic V Cicchetti. 1976. “Sleep in Mammals: Ecological and Constitutional Correlates.” *Science* 194 (4266): 732–34.

Ambrose, Susan A, Michael B Bridges, Michele DiPietro, Marsha C Lovett, and Marie K Norman. 2010. *How Learning Works.* Jossey-Bass.

Anderson, L. W., and D. R. Krathwohl. 2001. *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom’s Taxonomy of Educational Objectives.* Longman.

Anderson, Neil, Maria Angela Ferrario, Aidan McGowan, et al. 2025. “Learning to ‘Think’ Through Playful Interactions: A Play-Kit for Incoming First-Year Computing Students.” *2025 IEEE Global Engineering Education Conference (EDUCON)*, 1–3.

Antonio, Nuno, Ana de Almeida, and Luis Nunes. 2019. “Hotel Booking Demand Datasets.” *Data in Brief* 22: 41–49.

Auker, Linda A., and Erika L. Barthelmess. 2020a. “Teaching r in the Undergraduate Ecology Classroom: Approaches, Lessons Learned, and Recommendations.” *Ecosphere* 11 (4). https://doi.org/10.1002/ecs2.3060.

Auker, Linda A., and Erika L. Barthelmess. 2020b. “Teaching r in the Undergraduate Ecology Classroom: Approaches, Lessons Learned, and Recommendations.” *Ecosphere* 11 (4): e03060. https://doi.org/https://doi.org/10.1002/ecs2.3060.

Bailenson, J. N. 2021. *Nonverbal Overload: A Theoretical Argument for the Causes of Zoom*

Fatigue, Technology, Mind, and Behavior. American Psychological Association.

Balreira, Dennis G., Thiago L. T. da Silveira, and Juliano A. Wickboldt. 2023. "Investigating the Impact of Adopting Python and C Languages for Introductory Engineering Programming Courses." *Computer Applications in Engineering Education* 31 (1): 47–62. https://doi.org/10.1002/cae.22570.

Beatty, B. 2019. *Hybrid-Flexible Course Design.* EdTech Books.

Bell, Tim, Jason Alexander, Isaac Freeman, and Mick Grimley. 2009. "Computer Science Unplugged: School Students Doing Real Computing Without Computers." *New Zealand Journal of Applied Computing and Information Technology* 13 (1): 20–29.

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.

Bezanson, Jeff, Alan Edelman, Stefan Karpinski, and Shah. Viral B. 2017. *Julia: A Fresh Approach to Numerical Computing.* 59: 65--98. https://doi.org/10.1137/141000671.

Biggs, John B., and Kevin F. Collis. 1982. *Evaluating the Quality of Learning: The SOLO Taxonomy (Structure of the Observed Learning Outcome).* Academic Press, Inc.

Bloom, Benjamin, Max D. Engelhart, Edward J. Furst, Walker H. Hill, and David R. Krathwohl. 1956. *Taxonomy of Educational Objectives: The Classification of Educational Goals.* Handbook 1: Cognitive Domain. Longman.

Blum, Susan D., ed. 2020. *Ungrading: Why Rating Students Undermines Learning (and What to Do Instead).* West Virginia University Press.

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.

Buckmiller, Tom, Randal Peters, and Jerrid Kruse. 2017. "Questioning Points and Percentages: Standards-Based Grading (SBG) in Higher Education." *College Teaching* 65 (4): 151–57. https://doi.org/10.1080/87567555.2017.1302919.

Carter, Janet, and Tony Jenkins. 1999. "Gender and Programming: What's Going On?" *Proceedings of the 4th Annual SIGCSE/SIGCUE ITiCSE Conference on Innovation and Technology in Computer Science Education* (Cracow Poland), June, 1–4. https://doi.org/10.1145/305786.305824.

Caspersen, Michael E., and Jens Bennedsen. 2007. "Instructional Design of a Programming Course: A Learning Theoretic Approach." *Proceedings of the Third International Workshop on Computing Education Research* (Atlanta Georgia USA), September 15, 111–22. https://doi.org/10.1145/1288580.1288595.

Centers for Disease Control and Prevention. n.d. *Behavioral Risk Factor Surveillance System (BRFSS) Data*. https://www.cdc.gov/brfss/annual_data/annual_data.htm.

Che-Castaldo, Judy P, Amy Byrne, Kaitlyn Perišin, and Lisa J Faust. 2019. "Sex-Specific Median Life Expectancies from Ex Situ Populations for 330 Animal Species." *Scientific Data* 6 (1): 190019.

Chin, Monica. n.d. *File Not Found: A Generation That Grew up with Google Is Forcing Professors to Rethink Their Lesson Plans (Internet Archive Link)*. The Verge. https://web.archive.org/web/20210922124725/https://www.theverge.com/22684730/students-file-folder-directory-structure-education-gen-z.

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.

Clark, David, and Robert Talbert. 2023. *Grading for Growth: A Guide to Alternative Grading Practices That Promote Authentic Student Learning and Student Engagement in Higher Education*. Routledge.

Colquhoun, Rebecca, Samantha Ahern, Brittany Blankinship, and Patricia Loto. 2026. *Teaching Programming Across Disciplines*.

Colquhoun, Rebecca, Tiago A. Marques, Brittany Blankinship, William Kay, Rob Young, and Ozan Evkaya. 2026. *Teaching Programming Across Disciplines*.

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.

Cooling, Chris, Nick Jayanth, Samantha Alvarez-Madrazo, Leila S. shafti, Joseph El Gemayel, and Lucia Michielin. 2026. *Teaching Programming Across Disciplines*.

Craske, Michelle G., Michael Treanor, Christopher C. Conway, Tomislav Zbozinek, and Bram Vervliet. 2014. "Maximizing Exposure Therapy: An Inhibitory Learning Approach." *Behaviour Research and Therapy* 58 (July): 10–23. https://doi.org/10.1016/j.brat.2014.04.006.

Curzon, Paul, Peter W McOwan, Nicola Plant, and Laura R Meagher. 2014. "Introducing

Teachers to Computational Thinking Using Unplugged Storytelling." *Proceedings of the 9th Workshop in Primary and Secondary Computing Education*, 89–92.

Cutting, D. 2025. *The ProgBoard*. https://thinklikeacomputer.org.

DeLeon, Abraham P. 2012. "'Anarchism... Is a Living Force Within Our Life...' Anarchism, Education and Alternative Possibilities." *Educational Studies* 48 (1): 5–11. https://doi.org/10.1080/00131946.2012.641849.

Denning, Peter J. 2017. "Remaining Trouble Spots with Computational Thinking." *Communications of the ACM* 60 (6): 33–39.

Desvages, Charlotte, Brittany Blankinship, Umberto Noè, and Pawel Orzechowski. 2026. *Teaching Programming Across Disciplines*.

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.

Freire, Paulo. 1996. *Pedagogy of the Oppressed*. Translated by Myra Bergman Ramos. Penguin.

Freire, Paulo. 2017. *Pedagogy of the Oppressed*. Penguin Modern Classics. Penguin Classics.

Gemayel, Joseph El, Arif Budiarto, and William Bell. 2026. *Teaching Programming Across Disciplines*.

Gimenez, Olivier, Fitsum Abadi, Jean-Yves Barnagaud, et al. 2013. "How Can Quantitative Ecology Be Attractive to Young Scientists? Balancing Computer/Desk Work with Fieldwork." *Animal Conservation* 16 (2): 134–36.

Goel, Sanjay, and Vanshi Kathuria. 2010. "A Novel Approach for Collaborative Pair Programming." *Journal of Information Technology Education: Research* 9 (1): 183–96.

Gorman, Kristen B, Tony D Williams, and William R Fraser. 2014. "Ecological Sexual Dimorphism and Environmental Variability Within a Community of Antarctic Penguins

(Genus Pygoscelis)." *PloS One* 9 (3): e90081.

Graeber, David. 2024. *Pirate Enlightenment, or the Real Libertalia*. Penguin Books.

Grizzle, Jessy. n.d. *Notes for Computational Linear Algebra*. GitHub. https://github.com/michiganrobotics/ROB-101-Textbook-Computational-Linear-Algebra/tree/main.

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

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." *Proceedings of the 25th Western Canadian Conference on Computing Education*, WCCCE '23, May, 1–7. 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.

Hannay, J. E., T. Dybå, E. Arisholm, and D. I. Sjøberg. 2009. "The Effectiveness of Pair Programming: A Meta-Analysis." *Information and Software Technology* 51 (7): 1110–22.

Harlfoxem. 2016. *House Sales in King County, USA*. Kaggle dataset. https://www.kaggle.com/datasets/harlfoxem/housesalesprediction.

Hoffbeck, Joseph P, Heather E Dillon, Robert J Albright, Wayne Lu, and Timothy A Doughty. 2016. "Teaching Programming in the Context of Solving Engineering Problems." *2016 IEEE Frontiers in Education Conference (FIE)*, 1–7.

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.

Johnson, Fionnuala, Stephen McQuistin, John O'Donnell, and Quintin Cutts. 2022. "Experience Report: Identifying Unexpected Programming Misconceptions with a Computer Systems Approach." *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1*, 325–30. https://doi.org/10.1145/3502718.3524775.

Kampmann, Isabel L., Paul M. G. Emmelkamp, and Nexhmedin Morina. 2018. "Does Exposure Therapy Lead to Changes in Attention Bias and Approach-Avoidance Bias in

Patients with Social Anxiety Disorder?" *Cognitive Therapy and Research* 42 (6): 856–66. https://doi.org/10.1007/s10608-018-9934-5.

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

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

King, Stuart, and Serveh Sharifi Far. 2025. "Teaching Data Science to Diverse Learners: A Hybrid and Interdisciplinary Approach." *Teaching Statistics*.

Kohn, Alfie. 1993. *Punished by Rewards: The Trouble with Gold Stars, Incentive Plans, A's, Praise, and Other Bribes.* Houghton, Mifflin and Company.

Koulouri, Theodora, Stanislao Lauria, and Robert D Macredie. 2014. "Teaching Introductory Programming: A Quantitative Evaluation of Different Approaches." *ACM Transactions on Computing Education (TOCE)* 14 (4): 1–28.

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.

Langtangen, Hans Petter. 2006. *Python Scripting for Computational Science*. Springer.

Larson, K. A., F. P. Trees, and D. S. Weaver. 2008. "Continuous Feedback Pedagogical Patterns." *Proceedings of the 15th Conference on Pattern Languages of Programs*, 1–14.

Learn Higher. 2012a. *Ground Rules for Group Work*. Association for Learning Development in Higher Education. https://aldinhe.ac.uk/product/learnhigher-resources/ground-rules-for-group-work/.

Learn Higher. 2012b. *Group Work Booklet*. Association for Learning Development in Higher Education. https://aldinhe.ac.uk/product/learnhigher-resources/group-work-booklet/.

Lewis, Drew. 2022. "Impacts of Standards-Based Grading on Students' Mindset and Test Anxiety." *Journal of the Scholarship of Teaching and Learning* 22 (2). https://doi.org/10.14434/josotl.v22i2.31308.

Lobb, Richard, and Jenny Harlow. 2016. "Coderunner: A Tool for Assessing Computer Programming Skills." *ACM Inroads* 7 (1): 47–51. https://doi.org/10.1145/2810041.

Luxton-Reilly, Andrew, Simon, Ibrahim Albluwi, et al. 2018. "Introductory Programming: A Systematic Literature Review." *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, 55–106.

Lye, Sze Yee, and Joyce Hwee Ling Koh. 2014. "Review on Teaching and Learning of Computational Thinking Through Programming: What Is Next for k-12?" *Computers in Human Behavior* 41: 51–61.

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).

Nilson, Linda B. 2014. *Specifications Grading: Restoring Rigor, Motivating Students, and Saving Faculty Time*. Stylus.

Nolan, Keith, and Susan Bergin. 2016. "The Role of Anxiety When Learning to Program: A Systematic Review of the Literature." *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.

O'Leary, Maura, and Richard Stockwell. 2021. "Skills-Based Grading: A Novel Approach to Teaching Formal Semantics." *Proceedings of the Linguistic Society of America* 6 (1): 869. https://doi.org/10.3765/plsa.v6i1.5025.

O'Leary, Maura, and Richard Stockwell. 2022. "Implementing Skills-Based Grading in a Linguistics Course." *American Speech* 97 (2): 247–62. https://doi.org/10.1215/00031283-

9940629.

Offutt, Jeff, Paul Ammann, Kinga Dobolyi, et al. 2017. “A Novel Self-Paced Model for Teaching Programming.” *Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale* (Cambridge Massachusetts USA), April, 177–80. https://doi.org/10.1145/3051457.3053978.

Oldnall, Chris, William Kay, Chris Sutherland, Tiago A. Marques, and Robert Young. 2026. *Teaching Programming Across Disciplines*.

Organisation for Economic Co-operation and Development. n.d. *PISA Dataset*. https://www.oecd.org/en/about/programmes/pisa/pisa-data.html.

Orzechowski, Pawel, Brittany Blankinship, and Kasia Banas. 2026. *Teaching Programming Across Disciplines*.

Orzechowski, Pawel, and Bea Alex Elaine Mowat Clare Llewellyn. 2026. *Teaching Programming Across Disciplines*.

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.

Papert, Seymour A. 2020b. *Mindstorms: Children, Computers, and Powerful Ideas*. Basic books.

Papert, Seymour A. 2020a. *Mindstorms: Children, Computers, and Powerful Ideas*. Basic books.

Park, M. J., T. Calistro, and A. A. Lipnevich. 2025. “Zooming to Work or Working to Zoom: Relations Between Zoom Fatigue, Cognitive Load, and Boredom.” *Research Papers in Education* 40 (6): 822–50. https://doi.org/10.1080/02671522.2025.2507592.

Parker, Priya. 2018. *The Art of Gathering*. Penguin.

Phillips, Katrina F, Gustavo D Stahelin, Ryan M Chabot, and Katherine L Mansfield. 2021. “Long-Term Trends in Marine Turtle Size at Maturity at an Important Atlantic Rookery.” *Ecosphere* 12 (7): e03631.

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.

Porter, L., C. Bailey Lee, B. Simon, Q. Cutts, and D. Zingaro. 2011. “Experience Report: A

Multi-Classroom Report on the Value of Peer Instruction." *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education*, 138–42.

Raes, A., L. Detienne, I. Windey, and F. Depaepe. 2020. "A Systematic Literature Review on Synchronous Hybrid Learning: Gaps Identified." *Learning Environments Research* 23 (3): 269–90.

Rea, Ashley. 2022. "Coding Equity: Social Justice and Computer Programming Literacy Education." *IEEE Transactions on Professional Communication* 65 (1): 87–103. https://doi.org/10.1109/TPC.2022.3143965.

Robins, Anthony, Janet Rountree, and Nathan Rountree. 2003. "Learning and Teaching Programming: A Review and Discussion." *Computer Science Education* 13 (2): 137–72.

Robinson, Chin Choo, and Hallett Hullinger. 2008. "New Benchmarks in Higher Education: Student Engagement in Online Learning." *Journal of Education for Business* 84 (2): 101–9. https://doi.org/10.3200/JOEB.84.2.101-109.

Sackstein, Starr. 2015. *Hacking Assessment: 10 Ways to Go Gradeless in a Traditional Grades School*. Vol. 3. Hack Learning. Times 10 Publications.

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

Sambell, Kay, Liz McDowell, and Catherine Montgomery. 2012. *Assessment for Learning in Higher Education*. Routledge.

Sarvary, Mark. 2014. "Biostatistics in the Classroom: Teaching Introductory Biology Students How to Use the Statistical Software 'r' Effectively." *Tested Studies for Laboratory Teaching Proceedings of the Association for Biology Laboratory Education* 35 (January): 405–7.

School, EDHEC Business. 2021. *Online Diplomas Versus MOOCs: Advantages and Downsides*. Https://online.edhec.edu/en/blog/online-diplomas-versus-moocs/[3].

Sentance, Sue, and Andrew Csizmadia. 2017. "Computing in the Curriculum: Challenges and Strategies from a Teacher's Perspective." *Education and Information Technologies* 22 (2): 469–95.

Shafti, Leila S., and Joseph El Gemayel. 2026. *Teaching Programming Across Disciplines*.

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

[3] https://online.edhec.edu/en/blog/online-diplomas-versus-moocs/

Singer, Jeremy. 2020. "Notes on Notebooks: Is Jupyter the Bringer of Jollity?" *Proceedings of the 2020 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software.* https://doi.org/10.1145/3426428.3426924.

Singer, Jeremy, and Steve Draper. 2025. "Let's Take Esoteric Programming Languages Seriously." *Proceedings of the 2025 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, Onward! '25, October, 213–26. https://doi.org/10.1145/3759429.3762632.

Skipsey, Sam, Gordon Stewart, Jeremy Singer, and David Cutting. 2026. *Teaching Programming Across Disciplines.*

Sklar, Julia. 2020. *'Zoom Fatigue' Is Taxing the Brain. Here's Why That Happens.* Https://www.nationalgeographic.com/science/article/coronavirus-zoom-fatigue-is-taxing-the-brain-here-is-why-that-happens[4].

Smith, Jack W, James E Everhart, William C Dickson, William C Knowler, and Robert Scott Johannes. 1988. "Using the ADAP Learning Algorithm to Forecast the Onset of Diabetes Mellitus." *Proceedings of the Annual Symposium on Computer Application in Medical Care*, 261.

Stewart, Graeme Andrew, Moreno Briceño, Alexander, Gras, Philippe, et al. 2025. "Julia in HEP." *EPJ Web Conf.* 337: 01266. https://doi.org/10.1051/epjconf/202533701266.

Stommel, Jesse. 2023. *Undoing the Grade: Why We Grade, and How to Stop.* Hybrid Pedagogy Inc.

Suissa, Judith. 2006. *Anarchism and Education: A Philosophical Perspective.* Routledge.

Sweller, John. 1988. "Cognitive Load During Problem Solving: Effects on Learning." *Cognitive Science* 12 (2): 257–85.

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

Tedre, Matti, and Peter J Denning. 2016. "The Long Quest for Computational Thinking." *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*, 120–29.

[4] https://www.nationalgeographic.com/science/article/coronavirus-zoom-fatigue-is-taxing-the-brain-here-is-why-that-happens

The World Happiness Report. n.d. Https://worldhappiness.report/[5].

Tshukudu, Ethel, Quintin Cutts, and Mary Ellen Foster. 2021. "Evaluating a Pedagogy for Improving Conceptual Transfer and Understanding in a Second Programming Language Learning Context." *Proceedings of the 21st Koli Calling International Conference on Computing Education Research.* https://doi.org/10.1145/3488042.3488050.

United Nations Educational, Scientific and Cultural Organization (UNESCO). 2023. *Global Education Monitoring Report 2023: Technology in Education – a Tool on Whose Terms?* UNESCO. https://www.unesco.org/gem-report/en/publication/technology.

University of Edinburgh. 2024. "Group Working." In *Institute for Academic Development.* https://institute-academic-development.ed.ac.uk/study-hub/learning-resources/group-working.

Watson, Christopher, and Frederick WB Li. 2014. "Failure Rates in Introductory Programming Revisited." *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education*, 39–44.

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

Whitton, Nicola. 2022. *Play and Learning in Adulthood: Reimagining Pedagogy and the Politics of Education.* Springer Nature.

Wilkinson, Ryan Gerald, and Connor Ashworth. 2025. "Fragments of Anarchism in Higher Education Critical Art Pedagogies." *Arts and Humanities in Higher Education.* https://doi.org/10.1177/14740222241313303.

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

Williams, L., R. R. Kessler, W. Cunningham, and R. Jeffries. 2000. "Strengthening the Case for Pair Programming." *IEEE Software* 17 (4): 19–25.

Wing, Jeannette M. 2006. "Computational Thinking." *Communications of the ACM* 49 (3): 33–35.

Wood, David Muir. 2012. *Civil Engineering: A Very Short Introduction.* Vol. 331. Oxford University Press.

Xiao, Pei, Liang Chen, Xiaoqin Dong, et al. 2022. "Anxiety, Depression, and Satisfaction with Life Among College Students in China: Nine Months After Initiation of the Outbreak

[5] https://worldhappiness.report/

of COVID-19." *Frontiers in Psychiatry* 12 (January). https://doi.org/10.3389/fpsyt.2021.777190.

Zarb, Mark, and Janet Hughes. 2015. "Breaking the Communication Barrier: Guidelines to Aid Communication Within Pair Programming." *Computer Science Education* 25 (2): 120–51.

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.

Zuraw, Kie, Ann M. Aly, Isabelle Lin, and Adam J. Royer. 2019. "Gotta Catch 'Em All: Skills Grading in Undergraduate Linguistics." *Language* 95 (4): e406–27. https://doi.org/10.1353/lan.2019.0081.