Practices To Foster Inclusion and Accessibility in Programming Teaching

Rebecca L. Colquhoun

Department of Earth Science and Engineering, Imperial College London
r.colquhoun23@imperial.ac.uk

Samantha Ahern

Centre for Advanced Research Computing, UCL
s.ahern@ucl.ac.uk

Gule Saman

School of Engineering & Physical Sciences, Heriott-Watt University
s.gule@hw.ac.uk

Brittany Blankinship

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

Patricia A. Loto

Software Quality Research Group, Universidad Nacional del Nordeste(UNNE)
patricialoto@comunidad.unne.edu.ar

Introduction

We believe that learning programming should be a truly equitable and inclusive process, in which all students are able to take part and succeed. In order to achieve this, we — as programming educators — must continually and critically assess the fundamentals of our teaching techniques. In this chapter, we will first briefly discuss what makes teaching accessible or inaccessible for different groups of students (including but not limited to disabled students and students learning in an additional language), before reviewing different pedagogical approaches often used in teaching programming in light of these principles. These reflections are intended to empower educators to adapt and innovate rather than to act as a prescriptive guidebook.

There are three main aspects to consider when making programming teaching accessible: 1. making the workspace accessible, 2. making the instruction accessible, and 3. considering the implications of the method of delivery. Most universities have a student disability office, who can provide advice on making course materials accessible and will often help students to gain access to assistive technology software. Advice might include making sure to use tagged PDFs or Markdown, or whichever typeface, size and colour combinations are best for use on a presentation. However, these offices may be less familiar with helping students to access programming environments. For some students, simply changing the editor theme will be sufficient. Others may need to experiment with different editors to find one which works with their assistive technology (see e.g. Kramer (2024)). We will not discuss this any further here, but instead refer readers to resources such as Hadwen-Bennett, Sentance, and Morrison (2018).

In this chapter, we focus on the implications of methods of delivery to the accessibility of programming teaching. Along the way, we identify where adaptations to a particular method might make it more suitable or accessible for a certain group of learners. The pedagogical framework that we use (knowingly or unknowingly) to organise our teaching sessions can impact the accessibility of our teaching: we can consider a continuum from transmissive learning (in which knowledge is “transmitted” to students by instructors e.g. through direct instruction) to student-led, discovery based learning (often based on a constructivist model). Programming is a subject which could lend itself well to authentic problem based learning wherein students construct their own knowledge. However, this can present additional barriers to neurodivergent students, who often experience a higher cognitive burden whilst completing discovery-based learning tasks compared to tasks where they were explicitly directed on what to focus on, and compared to their neurotypical peers (Gabay et al. 2025). Problem solving, which is inherent in discovery based learning, places a large burden on working memory (Sweller 1988) which neurodivergent individuals may have weaknesses in (Maehler and Schuchardt 2016). Discovery based learning may also be more suited to expert learners, who have a bank of pre-existing knowledge to work from, than to novices (Kirschner, Sweller, and Clark 2006). This illustrates how teaching using a different pedagogic framework can be accessible for different groups of learners.

Some students may find it hard to generalise, or to engage when examples are not relevant to their chosen domain. Consider gathering feedback on student interests, learning preferences, and backgrounds through a pre-course survey, in order to curate relevant examples and case studies that resonate with student experiences (Kahu, Nelson, and Picton 2017). Engaging students in building course content signals that their opinions shape the course content, fostering a sense of community and ownership. Pre-course surveys also give another opportunity for students to disclose any disabilities or access needs. Some students may not have access to a suitable device either for in class or out

of class activities. Asking about device access in a pre-course survey can help to find solutions (e.g. a loaner laptop or finding a combination of software which works with any assistive technology) before classes begin.

A key aspect of making content accessible is considering cognitive load. In this, the cognitive engagement a task requires can be divided into 'intrinsic load' (the inherent difficulty of the task), 'germane load' (the work put into creating a permanent store of knowledge) and 'extraneous load' (the way information is presented) (Sweller 1988; Paas, Renkl, and Sweller 2003). Everyone has a limit to their working memory, and when the total cognitive load exceeds this, learning is impacted. The germane and extraneous load of a task can vary between learners (for example due to neurodivergence (Le Cunff, Giampietro, and Dommett 2024) or due to learning in an additional language) and is also influenced by how material is presented. In the chapter, we discuss some ways that educators can manage the extraneous load in their classrooms, to ensure that students have capacity to engage in the germane load of building long-term mental schemas of programming logic (Sweller 1994).

Generally, designing teaching sessions and materials around Universal Design for Learning (UDL) principles helps maximise accessibility. UDL centers on students having agency and designing materials to have multiple options for representation, engagement and action (CAST 2024).

Ordering of Material

In teaching programming there are different approaches to how material is presented. For example, Selby identifies four approaches: 1. code analysis, 2. building blocks, 3. simple units and 4. full systems (Selby 2011). Code analysis begins with students analysing and understanding existing code before going onto writing pseudocode. Only later do students write full programs. This approach provides a structured way for students to first focus on programming logic before having to worry about syntax, which may reduce cognitive load. However some students may struggle with knowledge transfer between writing pseudocode and writing compilable code (Glaser, Hartel, and Garratt 2000).

In other approaches (building blocks and simple units), students learn small units of code (individual constructs or set coding "phrases" respectively) and gradually build up to writing longer programs. These can lead to fragmented understanding, where students are not sure how to combine the components introduced together, and the early focus on syntax may be overwhelming for some (Selby 2011). However, it does allow students to build up a toolbox of reusable code, which they can then apply in other scenarios, reducing the amount of recall and typing of precise syntax which is required which may pose a challenge for some students. The final approach that Selby (2011) outlines is full systems, where students are immersed in a real-life problem and programming concepts are introduced as needed. This is close to the discovery-led learning discussed previously.

Classroom Models

These different approaches to teaching programming can be delivered in different ways. We have identified six different classroom models, though there is significant overlap and they are not mutually exclusive. We will now discuss each approach in turn, identifying what makes them accessible, and where they may present an accessibility barrier. Where possible, we identify possible modifications to the model to increase accessibility.

Participatory Live Coding

Live coding involves an instructor writing code live at the front of the classroom, whilst narrating what they are doing. It becomes participatory by students following along, writing and executing code at the same time (Nederbragt et al. 2020). Participatory live coding has many advantages, hence why it forms a key tenet of the Carpentries approach to teaching coding (The Carpentries 2025). For many students, watching someone code and following along is far more engaging than watching someone present a slide deck. In particular, it allows students the chance to ask questions and for the instructor to respond and adapt the teaching accordingly.

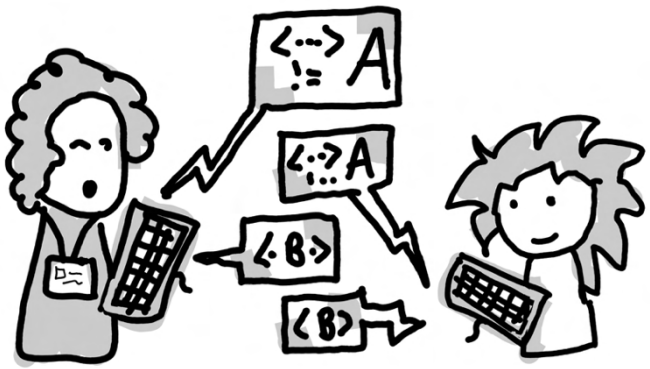

Figure 13.1: I code it. Then you code it. Repeat.

The instructor needing to type out code also reduces the pace of instruction: it is all too easy to quickly flash up some code on a slide, and have moved onto the next slide before the student has been able to write it down, which is a problem if slide decks are not provided in advance. Equally, it helps keep students on track. The lesson progresses at a set pace reducing the opportunity for students to take exit ramps or get lost on an unrelated tangent. However, the set pace can be an accessibility barrier for some, as it still depends on the instructor being able to judge an appropriate pace that will not leave students behind. Some students may require a slower pace

than others, for example dyslexic students may struggle to accurately copy down material from the screen and so will need more time to do this (Blampain, Gosse, and Van Reybroeck 2021).

As well as students requiring different amounts of time to copy down information from the board, some may find this difficult or impossible to do. The participatory live coding approach is particularly challenging for students who use a screen-reader, since they would have to balance auditory input from their computer with the instructor talking. To modify the approach for students who cannot see the board or accurately transcribe material, the instructor could provide code snippets or help people follow along using a git repository and tags (Bostian 2019; Sutton 2018). A collaborative notes document may be another approach. These ideas reduce the need for students to copy from the board, instead following along with small code chunks at a time. Students are also aided by the instructor making sure they verbally describe the actions and perhaps even dictating each word they type. This helps students follow along without needing to look up for each word typed.

In a participatory live coding approach, students also benefit from watching how the instructor codes: they are able to model the approach to a problem and the making and solving of mistakes. Explicitly modelling problem solving is particularly helpful for neurodivergent or anxious students for whom arriving at an error, mistake or block may become a barrier for learning rather than an opportunity.

Some students may benefit from switching to a flipped classroom approach where the instructor is pre-recorded doing a small chunk of coding (see *A Video Introduction to Live Coding Part 2* (2021) for an example of a single chunk), and students can follow along. This could either be done in the classroom, with instructors on hand to help students as needed, or be done at home and the classroom time be dedicated to longer problems. This approach also allows for students to turn on captions if needed.

Alternatively, the session could be live streamed, so students can see the projected screen more easily on their own computers, adjust the volume and can turn on captions if they wish, whilst still being able to ask questions.

Pair Programming

Instructors may want to include time where students independently solve programming problems, either as part of a broader lesson, or as part of inquiry based learning. One way to do this is through pair-programming, where students work in pairs to solve a problem, each taking turns to be the 'driver' and the 'navigator'. This concept is discussed in more detail in another chapter in this book (Desvages et al. 2026). Pair programming has been found to lead to better outcomes in introductory programming courses compared to courses where students program alone (Hanks et al. 2011; Faja 2014).

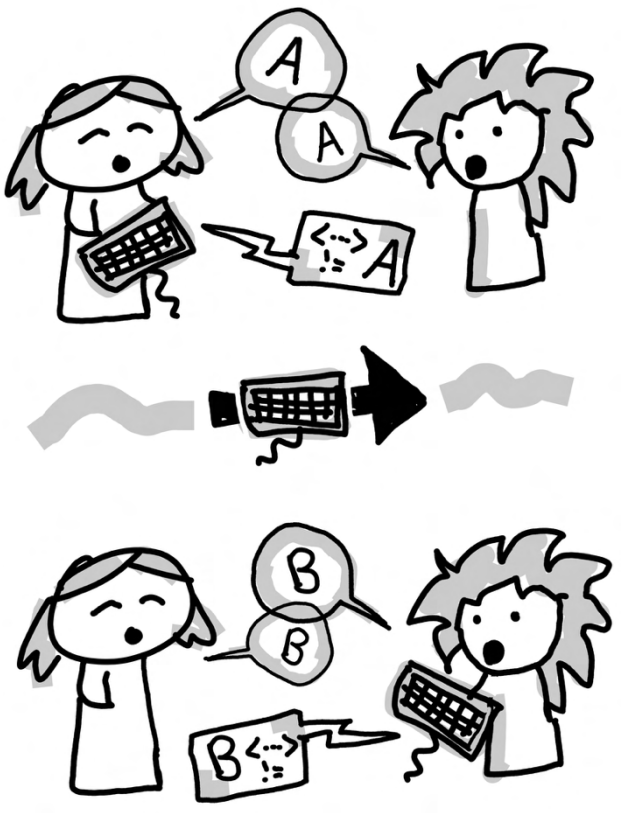

Figure 13.2: I drive, you navigate. Now let's switch who's driving.

A key advantage of pair programming is that it allows students to see different approaches to solving the same problem. For weaker or less confident students, this can help them build confidence and can model how to break down a problem. However, for others, students may compare themselves negatively to one another. One way to alleviate this is to switch partners around, which allows students to see different strengths and weaknesses in coding. For students who struggle to keep focussed, pair programming can be helpful, since both students are required to maintain active engagement with the task, frequently switching roles. It can also help keep students on task and working at the right pace, if the pairs are chosen carefully.

Pair programming can also serve to reduce the cognitive burden on students (Tsai, Yang, and Chang 2015) by allowing students to distribute tasks of high and low cognitive burden amongst themselves (Sands 2019; Winkler and Flatscher 2023). In classrooms where students have varying levels of experience with programming, pairs could consist of an expert and a novice where the expert (who would ordinarily experience lower extraneous load due to familiarity e.g. with the programming environment) can take on more of the extraneous tasks, allowing the novice to have more capacity for germane cognitive load.

A 2023 study from the University of Toronto Scarborough (Ali and Joordens 2023) found that there has been an increase in social anxiety amongst both higher education students and adolescents since the COVID-19 pandemic. Socially anxious students can experience anticipatory anxiety, impacting their abilities to communicate effectively with others and negatively impacting on their cognitive and emotional engagement. This could be reduced through the use of small-group icebreakers in sessions

prior to the introduction of pair programming. These activities help build a sense of familiarity and community within the learning environment, potentially reducing levels of anxiety (Archbell and Coplan 2022).

Whilst pair programming can reduce some of the cognitive load of programming, it can also introduce new sources of extraneous cognitive load since students now need to manage the social interaction associated with group work. It is therefore important to scaffold the task and provide a framework for how students should give feedback or interact during the exercise. This could look like modelling pair programming as part of the 'lecture' or by clearly explaining what will happen during the task. It should also include clearly signposting when switches in role and task will happen. Students may also try to impose one way of solving a problem as the 'correct' way, making it difficult for (neurodivergent) students who think differently, and expectations around this should be set during the task introduction.

Pair programming is often thought of as two people using one computer, switching over who is physically typing on the keyboard. However, students may find it difficult to use a computer with someone else's accessibility settings. It can therefore be helpful to use a shared workspace but on separate computers, so long as it is clear who is 'driving' at any given time. Students may need to balance working on the screen with auditory input which may be difficult for D/deaf or hard of hearing students or those who use a screen-reader. Some students may need their pair to work in a separate room to reduce background noise, which may be easier if pair-programming is being done virtually (e.g. through the use of breakout rooms).

Explicitly Modelling Problem Solving

When learning to program, whether in a classroom setting or independently, it is essential to recognise that programming is fundamentally about problem-solving and critical thinking as discussed by Govender (Govender 2007). The syntax of a programming language varies between languages, but what truly matters is developing the ability to think systematically about a problem and design an effective solution. Explicit modelling of the different stages in problem solving can be helpful for many students.

Figure 13.3: These are the steps I would take to solve this problem.

A helpful analogy is preparing a cup of tea. There are countless variations: some people prefer milk, others like sugar; some want herbal, others like it strong. The method you use depends on the preferences of the person you are making it for and any constraints involved (available ingredients, desired strength, etc.). You begin by understanding the requirements, clarifying constraints, and considering expectations, then decide on the steps to achieve the desired result. Programming is similar: syntax is just the final step in communicating a solution, but the real issue is clear thinking and structured planning.

The process of problem solving can be broken down into several steps. Some students will automatically do this, whilst others (particularly those who struggle with executive dysfunction) will benefit from modelling of the different stages and being provided with a checklist to follow:

- Understanding the problem
- Identifying constraints
- Identifying the expected inputs
- Considering the target audience or users
- Clarifying expectations
- Knowing about the desired outcomes (outputs)
- Designing the solution
- Testing possible solutions
- Troubleshooting and refining the approach

By foregrounding problem-solving instead of syntax, students can focus on creative and strategic thinking, which lowers barriers for those who might lack confidence with coding rules. However, not all learners will find abstract planning easy; some thrive by diving straight into code. For these learners, scaffolding the design stage can be helpful. Progressing from checklist to pseudocode, flowchart, and code supports learners with diverse preferences and reinforces both problem solving and implementation skills.

Accessible and inclusive programming education goes far beyond teaching language syntax. Focusing on a structured problem-solving process supports learners in developing analytical thinking, creativity, and confidence. By integrating scaffolding tools such as flowcharts, pseudocode, and concrete examples, educators can create a classroom environment where all students feel equipped to participate and succeed. Encouraging learners to think critically about problems, consider target audience and constraints, and work systematically before coding is important.

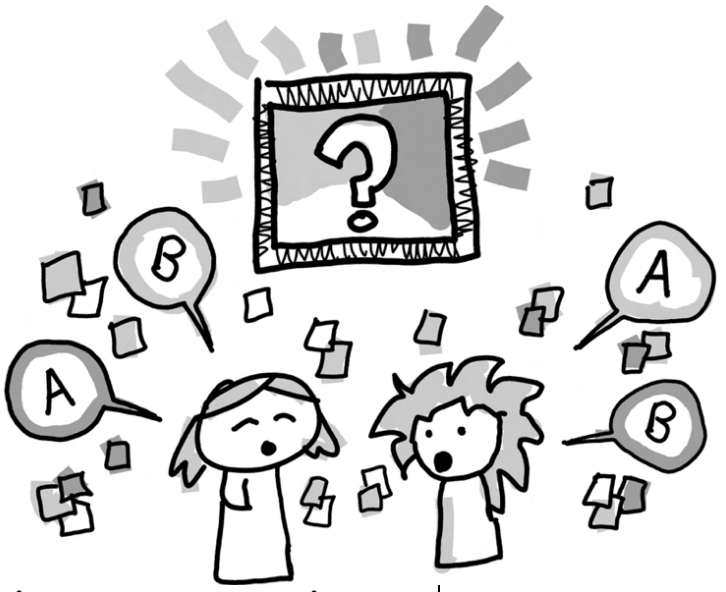

Figure 13.4: We learn new things when needed as part of a bigger project.

Discovery Based Learning

As we discussed in the introduction, instructors may prefer to use a discovery-based learning approach rather than the more direct-instruction that could lend itself well to participatory live coding. We previously discussed the cognitive load inherent in discovery based learning, but here consider other aspects of accessibility for this method of teaching.

The advantage of discovery based learning is that students have a reason for everything they learn. When students arrive at a problem, or need a new tool, they discover what is most suitable, and this can be more motivating to some students. However, some students may be overwhelmed by the scale of the task and struggle to break down a problem into manageable chunks. Part of the solution to this is making sure students are comfortable asking questions, rather than feeling like they have been 'abandoned' to teach themselves. This can be aided by having enough instructors and offering less obtrusive ways to ask questions, for example using a webchat or post-it note flag system. It may also be helpful to combine discovery based learning with some modelling of problem solving in a larger group. Another approach is to combine discovery based learning with pair-programming, with the inherent advantages and disadvantages we previously discussed. Finally, some students may struggle with the generalisation required and transfer to long-term memory: they may successfully identify a tool as suitable for this problem, but when it comes to the next problem they may start from scratch rather than being able to apply their existing knowledge. Part of this is developed with repeated exposure to different problems, but prompts from instructors may also help students realise how ideas can be generalised.

'Traditional' Lecture and Lab Sessions

The 'traditional' model of a lecture with a lab/practical session (e.g. Prather and Schlesinger 1978) combines instructor-led teaching with an opportunity to partake in practice of the material and/or discovery based learning. However, in the lecture session concepts or ideas may be introduced at a pace where some students become anxious about trying to process large amounts of content in real-time. This can be exacerbated by the use of dismissive language such as "just" or "even a monkey can do...". In addition, neurodivergent students can struggle to retain focus when there is a lot of information or lack of interaction.

The accessibility of lecture sessions can be improved by providing the session materials in advance. This enables students to engage with them at their own pace and identify any questions they may have for the lecture session. In addition, the inclusion of interactive components in the lecture session such as anonymous quizzes or discussions, enables reinforcement of learning without potentially causing discomfort amongst neurodivergent learners. However, it is important to be mindful that too much interaction may lead to cognitive overload. The regular use of lecture capture can make recordings available to students to review content as and when needed, reducing the risk of anxiety from a fear of not being able to keep up.

Figure 13.5: I talk, you listen. Then you do, I help.

When done well, the combination of lecture and lab sessions can help facilitate the navigation of the liminal space between a threshold concept being introduced and it being understood. Our learners, may not navigate this linearly, but will go through the stages of developing:

- an abstract (or theoretical) understanding of a concept;
- a concrete (or applied) understanding — the ability to write a computer program, for instance;
- the ability to go from an understanding of the abstract concept to software design or concrete implementation;
- an understanding of the rationale for learning and using the concept; and
- an understanding of how to apply the concept to new problems — problems beyond those given as homework or lab exercises (McCartney et al. 2009).

Knuth (1991) spoke about the close relationship between theory and practice as follows:

> "The main point I want to emphasize [...] is that both theory and practice can and should be present simultaneously. Theory and practice are not mutually exclusive; they are intimately connected. They live together and support each other." (Knuth 1991, 3)

The interaction between theory and practice can be promoted through careful design of lab instructions so they highlight key concepts and links that students should explore during the lab.

Some students may struggle with moving from theory to practice. They may make a sequence of unsuccessful attempts to implement, and iterate on, a solution, or get stuck in theory-oriented discussions without trying any of their ideas in practice. To alleviate this risk and support connections between theory and practice, an educator could intersperse references to manual and guidance materials within a task or problem sheet, ensuring relavance and ease of use for novice learners in particular. Instructors can also include reminders that students should save, compile and run their code regularaly, even if it is not yet working as intended (Eckerdal, Berglund, and Thuné 2024).

Flipped Classroom

An alternative approach to structuring learning is the flipped classroom model, wherein students engage with learning materials before class, and classroom time is used to engage in problem solving activities (Bergmann and Sams 2012; Advance HE, n.d.). By providing learning materials in advance (most commonly through pre-recorded videos, although this may also be in the form of text or audio), students can engage at their own pace. For some students, removing any time pressure enables them to better engage with the material, since they can pause to take notes, take a break or consult other materials during the course of the 'lecture'. However, effective use of the flipped classroom requires students to engage with the material before class (Moraros et al. 2015; Gross et al. 2015; Forsey, Low, and Glance 2013) and some students may struggle with the lack of direction, either not engaging with the material or becoming distracted by tangents. In a flipped classroom model, it is particularly important to ensure that the structure and signposting of materials is clear, as this can easily become a source of confusion for students.

Having time between engaging with the flipped classroom materials and the in-person session enables students to identify any questions they have in advance, and it is important that materials are released well in advance of synchronous sessions to allow for meaningful engagement. In order to further ensure that students feel comfortable to ask questions, an anonymous question box could be introduced, which is used to guide the in-person sessions. However, if a student misunderstands something early on in the asynchronous teaching, this may lead them to misunderstand the rest of the topic, whereas they may have normally been able to clarify their understanding more immediately.

Students can engage with the asynchronous portion in a location they feel most comfortable. For some students with disabilities, traditional lecture theaters may not be accessible, for example because of the fluorescent lighting or seating options. However, whilst the lecture portion may be more accessible using a flipped classroom model, the practical portion may be less accessible, as they are often centered on working as a group. Group activities have been shown to not necessarily result in long-term learning for neurodivergent students, and they may prove overwhelming or impose an additional cognitive burden on some students (Durgungoz and Durgungoz

Figure 13.6: You watch and read at home. Later, we do and discuss in class.

2025). Some students, particularly those who are less confident in the course materials may also feel unprepared to engage in group discussions, if they have not fully understood the material for example.

One way to support students in group activities is to center discussions around artefacts that students have prepared beforehand such as a solution to a coding challenge (Dusek et al. 2018). This helps students prepare for likely topics of discussion beforehand, and could be a motivator for some students to engage more diligently in the asynchronous work. Even if it is not desirable for students to complete all the practical work beforehand, it can be advantageous to provide practice exercises to help students prepare for the interactive session to both help students prepare and let them know what to expect.

Framing either in-class or preparatory practical exercises as Parson's problems or as part of a series of 'fading' activities may also be beneficial. In some settings, students could have the opportunity to engage in the development of an artefact via coding challenges with the in-person session becomes the lab session, as described above. If these are designed well, they may incorporate components of pair programming, and stimulate deep learning through the integration of theory and practice with the support of the module staff. Providing material online and in advance provides an opportunity to ensure materials are accessible. For example, it is important to make sure that any recorded lectures or demonstrations have captions available, and that there are image descriptions and audio description of what is displayed on screen. Resources such as the STEMM baseline accessibility guidance are a good starting place (Tyson 2022).

Conclusion

There is no singular approach to teaching programming that will always be appropriate, regardless of whether you consider this from a pedagogical or accessibility standpoint. Here we have outlined six different approaches to teaching programming, and what attributes make them accessible, or not, to different groups of learners. The ideas presented are not meant to be exhaustive or prescriptive; indeed, some of the adaptations that we suggest for one method, may be precisely what you need to make a session accessible for a student even if you have made different pedagogical design choices. By outlining a range of approaches, we hope that this chapter will give educators the starting blocks they need to design sessions that give all their learners the best chance of success.

References

A Video Introduction to Live Coding Part 2. 2021. https://www.youtube.com/watch?v=SkPmwe_WjeY.

Advance HE. n.d. "Flipped Learning." https://www.advance-he.ac.uk/knowledge-hub/flipped-learning-0.

Ali, Hannah, and Steve Joordens. 2023. "Social Anxiety in University Students: Towards an Intentional Life-Skills Based Prevention Model." *Journal of Mental Health & Clinical Psychology* 7 (3): 19–36. https://doi.org/10.29245/2578-2959/2023/3.1282.

Archbell, Kristen A., and Robert J. Coplan. 2022. "Too Anxious to Talk: Social Anxiety, Academic Communication, and Students' Experiences in Higher Education." *Journal of Emotional and Behavioral Disorders* 30 (4): 273–86. https://doi.org/10.1177/10634266211060079.

Bergmann, Jonathan, and Aaron Sams. 2012. *Flip Your Classroom: Reach Every Student in Every Class Every Day*. 1st ed. Eugene, Or Alexandria, Va: International Society for Technology in Education.

Blampain, Elise, Claire Gosse, and Marie Van Reybroeck. 2021. "Copying Skills in Children with and Without Dyslexia." *Reading and Writing* 34 (4): 859–85. https://doi.org/10.1007/s11145-020-10094-6.

Bostian, Emma. 2019. "Using Git Tags to Version Coding Tutorials. Dev.to." March 16, 2019. https://dev.to/emmabostian/using-git-tags-to-version-coding-tutorials-39cc.

CAST. 2024. "Universal Design for Learning Guidelines Version 3.0. CAST UDL Guidelines." 2024. https://udlguidelines.cast.org.

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

Durgungoz, Fatma Canan, and Ahmet Durgungoz. 2025. "'Interactive Lessons Are Great, but Too Much Is Too Much': Hearing Out Neurodivergent Students, Universal Design for Learning and the Case for Integrating More Anonymous Technology in Higher Education." *Higher Education*, January. https://doi.org/10.1007/s10734-024-01389-6.

Dusek, Jeff, Daniela Faas, Emily Ferrier, Robyn Goodner, Alisha Sarang-Sieminski, Adva Waranyuwat, and Alison Wood. 2018. "Proactive Inclusion of Neurodiverse Learning Styles in Project-Based Learning: A Call for Action." In *2018 ASEE Annual Conference & Exposition Proceedings*, 30891. Salt Lake City, Utah: ASEE Conferences. https://doi.org/10.18260/1-2--30891.

Eckerdal, Anna, Anders Berglund, and Michael Thuné. 2024. "Learning

Programming Practice and Programming Theory in the Computer Laboratory." *European Journal of Engineering Education* 49 (2): 330–47. https://doi.org/10.1080/03043797.2023.2294953.

Faja, Silvana. 2014. "Evaluating Effectiveness of Pair Programming as a Teaching Tool in Programming Courses." *Information Systems Education Journal* 12 (6): 36–44.

Forsey, Martin, Mitchell Low, and David Glance. 2013. "Flipping the Sociology Classroom: Towards a Practice of Online Pedagogy." *Journal of Sociology* 49 (4): 471–85. https://doi.org/10.1177/1440783313504059.

Gabay, Yafit, Lana Jacob, Atil Mansour, and Uri Hertz. 2025. "Computational Markers Show Specific Deficits for Dyslexia and ADHD in Complex Learning Settings." *Npj Science of Learning* 10 (1): 38. https://doi.org/10.1038/s41539-025-00323-4.

Glaser, Hugh, Pieter H. Hartel, and Paul W. Garratt. 2000. "Programming by Numbers: A Programming Method for Novices." *The Computer Journal* 43 (4): 252–65. https://doi.org/10.1093/comjnl/43.4.252.

Govender, I. 2007. "Experiences of Learning and Teaching: Problem Solving in Computer Programming." *African Journal of Research in Mathematics, Science and Technology Education* 11 (2): 39–50. https://doi.org/10.1080/10288457.2007.10740620.

Gross, David, Evava S. Pietri, Gordon Anderson, Karin Moyano-Camihort, and Mark J. Graham. 2015. "Increased Preclass Preparation Underlies Student Outcome Improvement in the Flipped Classroom." Edited by Mary Lee Ledbetter. *CBE–Life Sciences Education* 14 (4): ar36. https://doi.org/10.1187/cbe.15-02-0040.

Hadwen-Bennett, Alex, Sue Sentance, and Cecily Morrison. 2018. "Making Programming Accessible to Learners with Visual Impairments: A Literature Review." *International Journal of Computer Science Education in Schools* 2 (2): 3–13. https://doi.org/10.21585/ijcses.v2i2.25.

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

Kahu, Ella, Karen Nelson, and Catherine Picton. 2017. "Student Interest as a Key Driver of Engagement for First Year Students." *Student Success* 8 (2): 55–66. https://doi.org/10.5204/SSJ.V8I2.379.

Kirschner, Paul A., John Sweller, and Richard E. Clark. 2006. "Why Minimal Guidance During Instruction Does Not Work: An Analysis of the Failure of Constructivist, Discovery, Problem-Based, Experiential, and Inquiry-Based Teaching." *Educational Psychologist* 41 (2): 75–86. https://doi.org/10.1207/s15326985ep4102_1.

Knuth, Donald E. 1991. "Theory and Practice." *Theoretical Computer Science* 90 (1): 1–15. https://doi.org/10.1016/0304-3975(91)90295-D.

Kramer, Nimrod. 2024. "10 Best Code Editors with Accessibility Features. Daily.dev." May 16, 2024. https://daily.dev/blog/10-best-code-editors-with-accessibility-features.

Le Cunff, Anne-Laure, Vincent Giampietro, and Eleanor Dommett. 2024. "Neurodiversity Positively Predicts Perceived Extraneous Load in Online Learning: A Quantitative Research Study." *Education Sciences* 14 (5): 516. https://doi.org/10.3390/educsci14050516.

Maehler, Claudia, and Kirsten Schuchardt. 2016. "Working Memory in Children with Specific Learning Disorders and/or Attention Deficits." *Learning and Individual Differences* 49 (July): 341–47. https://doi.org/10.1016/j.lindif.2016.05.007.

McCartney, Robert, Jonas Boustedt, Anna Eckerdal, Jan Erik Moström, Kate Sanders, Lynda Thomas, and Carol Zander. 2009. "Liminal Spaces and Learning Computing." *European Journal of Engineering Education* 34 (4): 383–91. https://doi.org/10.1080/03043790902989580.

Moraros, John, Adiba Islam, Stan Yu, Ryan Banow, and Barbara Schindelka. 2015. "Flipping for Success: Evaluating the Effectiveness of a Novel Teaching Approach in a Graduate Level Setting." *BMC Medical Education* 15 (1): 27. https://doi.org/10.1186/s12909-015-0317-2.

Nederbragt, Alexander, Rayna Michelle Harris, Alison Presmanes Hill, and Greg Wilson. 2020. "Ten Quick Tips for Teaching with Participatory Live Coding." Edited by Francis Ouellette. *PLOS Computational Biology* 16 (9): e1008090. https://doi.org/10.1371/journal.pcbi.1008090.

Paas, Fred, Alexander Renkl, and John Sweller. 2003. "Cognitive Load Theory and Instructional Design: Recent Developments." *Educational Psychologist* 38 (1): 1–4. https://doi.org/10.1207/S15326985EP3801_1.

Prather, Ronald E., and Judith D. Schlesinger. 1978. "A Lecture/Laboratory Approach to the First Course in Programming." *ACM SIGCSE Bulletin* 10 (1): 115–18. https://doi.org/10.1145/990654.990599.

Sands, Philip. 2019. "Addressing Cognitive Load in the Computer Science Classroom." *ACM Inroads* 10 (1): 44–51. https://doi.org/10.1145/3210577.

Selby, Cynthia. 2011. "Four Approaches to Teaching Programming." In.

Sutton, Marcy. 2018. "Live Coding Accessibility. Marcysutton.com." June 28, 2018. https://marcysutton.com/live-coding-accessibility.

Sweller, John. 1988. "Cognitive Load During Problem Solving: Effects on Learning." *Cognitive Science* 12 (2): 257–85.

https://doi.org/10.1207/s15516709cog1202_4.

———. 1994. “Cognitive Load Theory, Learning Difficulty, and Instructional Design.” *Learning and Instruction* 4 (4): 295–312. https://doi.org/10.1016/0959-4752(94)90003-5.

The Carpentries. 2025. “Live Coding Is a Skill. Instructor Training.” June 4, 2025. https://carpentries.github.io/instructor-training/17-live.html.

Tsai, Chia-Yin, Ya-Fei Yang, and Chih-Kai Chang. 2015. “Cognitive Load Comparison of Traditional and Distributed Pair Programming on Visual Programming Language.” In *2015 International Conference of Educational Innovation Through Technology (EITT)*, 143–46. Wuhan, China: IEEE. https://doi.org/10.1109/EITT.2015.37.

Tyson, Jim. 2022. “Accessible STEMM Introduction. Make Things Accessible.” October 21, 2022. https://www.makethingsaccessible.com/guides/accessible-stemm-introduction/.

Winkler, Till, and Rony G Flatscher. 2023. “Cognitive Load in Programming Education: Easing the Burden on Beginners with REXX.” In *Central European Conference on Information and Intelligent Systems*, 171–78. Faculty of Organization; Informatics Varazdin.