(Hello World)

THE MAGAZINE FOR COMPUTING & DIGITAL MAKING EDUCATORS

PROGRAMMING LANGUAGE TRANSFER
Guiding students from block- to text-based coding

PROGRAMMING THE FUTURE
Programming as an act of creativity

BEYOND THE VIBE
Reframing AI tools for coding friction

Issue 28 | Nov 2025 | helloworld.cc

PLUS **INCLUSIVE PROGRAMMING PEDAGOGIES** · PRIMM FOR YOUNG LEARNERS · **GRANT WRITING** · BEBRAS CHALLENGE · **ASTRO PI** · SPATIAL COMPUTING · **WHY DO WE TEACH COMPUTING?** · FUTURE CAREERS · **OFFLINE PROGRAMMING** · SEMANTIC WAVES · **EDUCATOR REFLECTIONS** · 'REEL' PROBLEMS · **AI UNPLUGGED ACTIVITIES** · RURAL AI LESSONS

Raspberry Pi Foundation

Experience CS

A free integrated curriculum for computer science

Experience CS empowers educators of elementary and middle school students (aged 8 to 14) to teach computer science through a standards-aligned curriculum that integrates CS concepts into core subjects like maths, science, languages, and the arts.

Created by educators for educators, Experience CS includes:

- Ready-to-use lesson plans, educator resources, and classroom materials.
- Creative projects using a version of Scratch built especially for schools.
- Simple and intuitive learning management features to track students' progress and manage classroom assignments.
- Professional development opportunities to help you feel confident teaching CS. No prior experience needed.

Discover more at: rpf.io/exp-cs-hw28

HELLO, WORLD!

We take for granted all the clever, creative programming that goes into the technology we use in our daily lives. But how do we best teach programming to support the next generation of innovators? And how do we support you, the amazing educators who are also trying to keep pace with the advancements in computer science, and who want to inspire their students?

We hope that Hello World can help with just that. The theme of this magazine issue — and of our new podcast mini series — is programming. This issue is packed with insightful research, practical advice, and thoughtful ways to best teach programming in your classroom.

In this issue, Simon Peyton Jones discusses programming the future, and how programming is really an act of creativity (pages 22–23). We also have an article from Ben Schafer and Sarah Diesburg, sharing reflections from K–12 teachers on their experience of learning and teaching programming (pages 42–45). And finally, Kala Grice-Dobbins shares her step-by-step guide for writing a successful grant application (pages 74–75), which can apply to anyone around the world.

We'd love to hear your thoughts on Hello World. Subscribers will soon receive our annual survey by email. Please take five to ten minutes to share how you use Hello World and how we can improve our magazine and podcast to better support your work. Hello World is created by educators, for educators, and your feedback helps us shape its future.

Meg Wang
Editor

FEATURED THIS ISSUE

MANNI CHEUNG

During his master's degree, Manni's research asked: what happens when students use AI to help them code, but they end up learning less? Read about his findings on pages 16–17.

LEONTAE GRAY WARD

Leontae is a STEM and computer science educator based in Indiana, USA. On pages 48–49, she discusses how CPD upgraded her approach to programming instruction.

ZACHARY FLOWER

Zachary shares three of his tried and tested unplugged activities to get students to rediscover the power of physical thinking in a digital world. Find out more on pages 52–53.

(HW)

Hello World is the official magazine of the Raspberry Pi Foundation

Raspberry Pi Foundation

EDITORIAL

Editor
Meg Wang

Contributing Editor
Dominick Sanders

Subeditors
Louise Richmond and Gemma Coleman

Subscriptions
Dan Ladbrook

Social Media
Sean Sayers

DESIGN

criticalmedia.co.uk

Designer
Dougal Matthews

Photography
The Raspberry Pi Foundation, Adobe Stock

Graphics
Rob Jervis

Cover
© MUTI

CONTRIBUTORS

James Abela, Sophie Ashford, Ross Barrett, Marc Berges, Neil Brown, Manni Cheung, Michael Conterio, Mark Crane, Andrew Csizmadia, Sethi De Clercq, Chidi Duru, Sarah Diesburg, Esther Mmbai Diera, Diane Dowling, Ryan Etheridge, Catherine Elliott, Zachary Flower, Tracy Gardner, Leontae Gray Ward, Kala Grice-Dobbins, Philippa Hanman, Justin Heath, Jeremy Hieb, Laura James, Simon Peyton Jones, Lauren Kisser, Michael Kölling, Annabel Lindner, Dorian Love, Thomas Mason, Michaela Mueller-Unterweger, Kaye North, Jigar Patel, Tess Ramsey, Ben Schafer, Sue Sentance, Crystal Sheldon, Bonnie Sheppard, Ethel Tshukudu, Pierre Weill-Tessier

Supported by ORACLE

Contributing Partner

bcs The Chartered Institute for IT Academy of Computing

CAS

This magazine is printed on paper sourced from sustainable forests and the printer operates an environmental management system which has been assessed as conforming to ISO 14001.

Hello World is published by the Raspberry Pi Foundation, 37 Hills Road, Cambridge, CB2 1NT. The publisher, editor, and contributors accept no responsibility in respect of any omissions or errors relating to skills, products, or services referred to in the magazine. Except where otherwise noted, content in this magazine is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 Unported (CC BY-NC-SA 4.0).

CC BY NC SA

recycle

(HW) CONTENTS

22–49 COVER FEATURES

PROGRAMMING
What is the future of programming, and how can educators best teach it?

NEWS, FEATURES, AND OPINION

6 NEWS
Astro Pi; celebrating Coolest Projects 2025; Bebras Challenge 2025; ten years of Moonhack

14 INCLUSIVE PROGRAMMING PEDAGOGIES
Effective teaching methods to include learners with additional needs

16 BEYOND THE VIBE
How AI tools might undermine learning unless we design for friction

18 FLARE: A FRAMEWORK FOR LEARNING ABOUT RELATIONAL ELEMENTS
Code comprehension through familiar strategies

20 WHY DO WE TEACH COMPUTING?
Computing education research identifies four main perspectives on what CS education is and what it is for

22 PROGRAMMING THE FUTURE
What programming is and where it's going in the age of AI

24 BEGINNING WITH SCRATCH
A new instructor's joy in learning to program in Scratch

26 FROM BLOCK- TO TEXT-BASED CODING
What programming language transfer teaches us about helping students to transition

29 STRYPE PROGRAMMING ENVIRONMENT
Simplifying Python for beginners with intuitive frame-based editing and interactive graphics

34 JAVA'S LATEST FEATURES
Why every student should know how to use Java to unlock their future

36 A-LEVEL PROJECTS
How to support students when they choose a programming language for their A-level project

38 PRIMM AND PROPER VIBE CODING
What happens when students bring ChatGPT, GitHub, and Canva into the classroom?

42 TEACHING AND LEARNING PROGRAMMING
Educator reflections on learning to program while teaching programming to students

46 INTEGRATING GENAI
An introduction to programming course which asks how we should work with, and not against, generative AI

48 BIT BY BIT
How CPD upgraded an educator's approach to programming instruction

32 **A HISTORY OF PROGRAMMING LANGUAGES**
The use of programming languages in UK CS education and the global impact of Python

40 **OFFLINE PROGRAMMING**
How every student can learn AI concepts regardless of access to the internet

54 **THE 'REEL' PROBLEM IN CLASSROOMS**
How short-form video apps are reshaping student attention and engagement in computing classrooms

50 **FUTURE CAREERS**
How educators can help their students prepare for industry with Amazon Future Engineer UK

52 **NO SCREENS, NO PROBLEM**
Rediscovering the power of physical thinking in a digital world

58 **RAISING CODERS IN THE AGE OF AI**
In a rural part of the USA, a family uses AI as a tool for coding, problem-solving, and meaningful projects

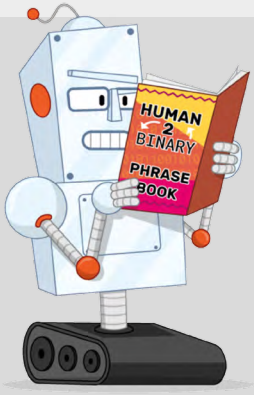
60 **BRIDGING THE ABSTRACTION GAP**
Using semantic waves to help scaffold problem-based learning in computer science education

62 **THE THRILL OF THE JOURNEY IN CODING**
Tips and activities to introduce the PRIMM approach for young digital explorers

83 SUBSCRIBE

LEARNING

RESOURCES & LESSON PLANS



66 **SEQUENCING**
A lesson plan about understanding the precise nature of the instructions that computers need to execute

70 **DEMYSTIFYING AI**
An open educational resource collection of unplugged AI teaching activities

CONVERSATION

56 **CODE EDITOR FOR EDUCATION**
Two teachers tell us about how they use our text-based coding environment in the classroom

74 **GRANT WRITING**
How to unlock funding for your classroom

76 **MEET THE CODE CLUB**
A conversation with a Code Club for Deaf creators in Nigeria

78 **SELF-PORTRAITURE THROUGH CODE**
Embracing artistry in computer science education

79 **BEBRAS**
A fun computational thinking challenge

80 **EVOLVING COMPUTING**
What is spatial computing?

82 **START AS YOU MEAN TO GO ON**
Tips for a strong start to the school year

ASTRO PI: INSPIRING YOUNG CREATORS FOR A DECADE OF DISCOVERY

Explore how the challenge has evolved over the past ten years into a STEM success story across Europe and beyond

Philippa Hanman

In 2015, a bold and brilliant idea launched into orbit (literally!). The Astro Pi Challenge, a European Space Agency (ESA) Education project run in collaboration with the Raspberry Pi Foundation, set out to inspire the next generation of programmers and space scientists by giving them the unique opportunity to write code that runs aboard the International Space Station (ISS). Now, ten years on, we celebrate a decade of creativity and education that has reached thousands of young people and their mentors across ESA member states.

Sophie Adenot, ESA astronaut, is our Astro Pi ambassador

A mission to expand coding capabilities

The European Astro Pi Challenge, aimed at enhancing computing and digital making skills among young people, originated as the Astro Pi UK competition with the UK Space Agency. Specially modified Raspberry Pi computers with Sense HATs were delivered to the ISS, by British ESA astronaut Tim Peake during his 2015 Principia mission, and students' programs were run with support from the crew.

Building on its success, ESA expanded the concept to the entire European region, launching the first European Astro Pi Challenge in 2016/2017.

Mission Zero, introduced in 2017/2018 alongside the original Mission Space Lab, offered a simpler activity for beginners: programming an Astro Pi to sense ISS temperature and display a message, all online without extra hardware. Its resources became available in 19 languages.

In 2021, new 'Mark II' Astro Pis were sent to the ISS, featuring two Raspberry Pi model 4s, a High-Quality Camera, and a custom Sense HAT with new sensors.

The 2022/23 Mission Zero shifted to a pixel art activity, where participants created 8x8 pixel images and animations using luminosity sensors to adjust background colours. A new online editor was introduced for direct code submission.

In 2023, Mission Space Lab moved away from distributing Astro Pi kits, instead providing the Astro Pi Replay tool, a simulated environment replaying ISS data and images. This gave young people unlimited access to unique earth observation images and data for testing their programs, captured by teams from previous years.

These missions aren't just educational. They cultivate problem-solving, collaboration, and a love for discovery to empower young people and their mentors. They also demonstrate how computer science and space exploration intersect in powerful and tangible ways.

Over the past ten years, the Astro Pi Challenge has engaged over 150,000 participants from more than 25 countries.

Celebrating impact: inspiring a generation

Since inception, the challenge has established itself as an opportunity for teachers to encourage students' enthusiasm for tech and team work. This has been demonstrated at St Joseph's School in Rush, Ireland, where there is a strong focus on making opportunities to engage with computing technologies accessible to all (**helloworld.cc/st-joseph**).

On hearing about this exciting coding challenge, teacher Danny introduced it to his computer science class, as well as extending an open invitation to all St Joseph's students. Lots of students signed up. Collaborating on Astro Pi projects has

Raspberry Pi Foundation esa

TAKE PART IN THIS MILESTONE YEAR: ASTROPI.ORG

enabled young people at St Joseph's to team up and uncover their strengths, and foster a strong community who have gone on to expand their digital skills beyond the curriculum, instilled with the belief that they can achieve anything.

The Astro Pi Challenge's tenth anniversary is not just a milestone, it's a testament to what happens when young people are given the tools and the trust to do extraordinary things. Across Europe and beyond, countless students have had their first taste of coding, data analysis, and scientific experimentation through this initiative. Many have gone on to pursue studies and careers in science, technology, engineering, and mathematics (STEM), inspired by the knowledge that their work reached outer space.

"The idea that your code could run in space is simply amazing. It's a fascinating and inspiring experience. The thought of sending a friendly message to astronauts and visitors aboard the ISS — a small gesture to connect Earth and space — it's an incredible feeling. Knowing that something I created might motivate other kids to explore coding and space is truly special." — Team Tony Pi

Educators, too, have found a powerful teaching tool in the Astro Pi Challenge. It brings together curriculum goals with real-world applications in a way few other projects can. Teachers often report that the challenge sparks enthusiasm in students who might otherwise feel disconnected from science or coding.

"Participating in Mission Space Labs offers students a great opportunity to work with the International Space Station, to see the Earth from above, to challenge them to overcome the terrestrial limits." — Mission Space Lab mentor

What can we expect from the 2025/26 Astro Pi Challenge?

This year we are excited to welcome ESA astronaut Sophie Adenot as our Astro Pi ambassador. She was awarded the French National Order of Merit (Chevalier) in 2022 and the medal of the French National Assembly honouring her actions as an inspiring ambassador for gender equality in sciences in 2021. She is scheduled to fly to the ISS, with the mission currently planned for 2026.

There are lots of helpful resources available, including step-by-step project guides, making it easy to engage your students in computer science and space exploration. Ten years ago, the idea of a school child's code running in space might have seemed like science fiction, yet today, it's an annual reality for thousands of students!

Both Mission Zero and Mission Space Lab are open for entries. Take part in this milestone year by visiting **astropi.org.** (HW)

IMPORTANT DATES

Astro Pi 2025/2026 is now open!

16 February 2026: closing date for Mission Space Lab

23 March 2026: closing date for Mission Zero

YOUNG MINDS, BIG IDEAS

A CELEBRATION OF COOLEST PROJECTS 2025

Every year, Coolest Projects celebrates young digital creators around the world with an online showcase and multiple in-person events

Sophie Ashford

Coolest Projects offers a platform for young people to celebrate a shared passion for computer science and get involved in STEM in a fun and inclusive environment. All projects are welcomed — from big to small, beginner to advanced, work in progress to finished creations.

For 2025, the Raspberry Pi Foundation organised an online showcase and in-person events in Ireland, the UK, and the USA. An in-person event was held in India for the first time, in Hyderabad this September.

These events were supported by partner-activated celebrations throughout the year, with more still scheduled for later in 2025. It was so great to see the community gathering all over the world, including in Sri Lanka, Belgium, South Africa.

The online showcase was celebrated with a livestream in June, championing the 11,980 participants from 41 countries who entered 5,952 tech projects between them. Each project demonstrated creativity and a commitment to learning.

Check out some of the highlights from what has been an incredible year for Coolest Projects. We look forward to returning in 2026 and continuing to celebrate young creators!

> EACH PROJECT DEMONSTRATED CREATIVITY AND A COMMITMENT TO LEARNING

Coolest Projects Ireland

In March, young creators gathered in Dublin for Coolest Projects Ireland. This event showcased the creativity and problem-solving skills of over 80 participants from across Northern Ireland and the Republic of Ireland.

Coolest Projects USA

In April, Coolest Projects USA took place at the Science Museum of Minnesota. Around 40 young people showcased their projects, sharing their work with friends, family, and the wider coding community. With hands-on tech activities and project demonstrations, the day was packed with energy.

Coolest Projects UK

At this event held in Bradford in May, over 170 young digital makers from across the UK gathered for Coolest Projects UK. Creators, families, mentors, and even some Scout troops joined in for a day of fun.

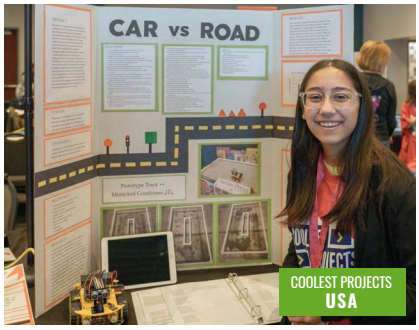
Global partners: spreading the Coolest Projects spirit

We're incredibly fortunate to collaborate with amazing partners worldwide. These dedicated teams are passionate about bringing the Coolest Projects spirit to their communities. From Indonesia to Belgium to Nigeria, creators globally are gathering, sharing ideas, and celebrating their creativity.

Thank you to our sponsors

Support from our Coolest Projects sponsors means we can make the online showcase and in-person events inspiring experiences for the young people taking part. We want to say a big thank you to our Champion Sponsors Broadcom Foundation, Allianz, Amazon Future Engineer, Meta, and Qube RT.

To learn more, visit **coolestprojects.org**. (HW)



BEBRAS CHALLENGE

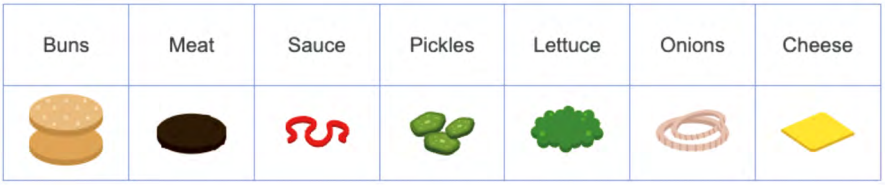
The Bebras Challenge is back for 2025, and is an engaging stepping stone to coding for your learners

Andrew Csizmadia

The Bebras Challenge (bebras.org) is recognised and acknowledged as the world's largest computing challenge with nearly 4 million learners in 85 countries participating in the last school year. In November 2024, over 467,000 learners participated in the UK Bebras Challenge (bebras.uk).

Connecting to key programming concepts

When Bebras tasks are created, they are classified by members of the international Bebras community against at least one of five computing domains. These domains agreed by the Bebras community are: Algorithms and Programming, Data Structures and Representations, Computer Processes and Hardware, Communications and Networking, and Interactions, Systems and Society. In a 2017 paper (**helloworld.cc/dagiene**), Dagienė, Sentance, and Stupurienė identified programming topics from Bebras tasks that members of the international Bebras community had designed, developed, refined, and possibly used in their own country's Bebras Challenge from 2004 to 2016. They identified 45 topics within the Algorithms and Programming domain (**Table 1**).



Now you try!

Let's explore some Bebras tasks that introduce or reinforce programming concepts.

■ **Figure 1** Burger Recipe task

Example 1: introducing programming concepts

The Burger Recipe task can be used as an introduction to the programming concepts of either a sequence of instructions or constraint checking.

Burger Recipe

Jessica is making burgers according to the following rules:

1. The sauce should be right above the meat
2. Meat and cheese should be below the pickles, lettuce, and onions
3. Onions should not be in contact with the buns

Question

Which burger in **Figure 1** is correctly made according to the rules?

In this task, learners are introduced to the concept of a constraint satisfaction problem, in which they have to find a solution which satisfies the constraints they are presented with. A constraint is a rule or limitation that restricts the possible values or configurations. In computer science, finding out whether a solution obeys all the given rules is called constraint checking.

Example 2: debugging

In the Maze Runner task, learners are introduced to the concept of debugging code which currently does not work correctly ('smelly code') to code that functions correctly.

Maze Runner

A robot has to move from the red square to the green square in **Figure 2**. If you press the Run button, you will see the program has an error!

Domain	Keywords
Algorithms and Programming	Algorithm; Binary search; Boolean algebra; Breadth-first search; Brute-force search; Bubble sort; Coding; Computational complexity; Constants; Constraints; Debugging; Depth-first search; Dijkstra's algorithm; Dynamic programming; Divide and conquer; Encapsulation; Function; Greedy algorithm; Heuristic; IF conditions; Inheritance; Iteration; Kruskal's algorithm; Logic gates; Loop; Maximum flow problem; Objects; Operations AND, OR, NOT; Optimisation; Parameters; Prim's algorithm; Procedure; Program; Programming language; Program execution; Quick sort; Recursion; RSA algorithm; Shortest path; Searching; Sorting; Travelling salesman problem; Variables.

■ **Table 1** Topics within the Algorithms and Programming domain of Bebras tasks

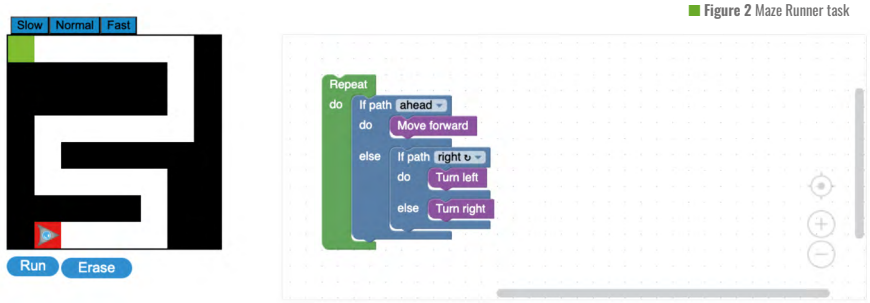


Figure 2 Maze Runner task

Task

Fix the program to get the robot to its green square.

In this task, learners are looking for a mistake in the code they have been given. This is called debugging, which is an important skill for computer programmers to develop.

Example 3: code optimisation

In the Shortest Program task, learners are encouraged to consider optimising code to make the program more efficient and execute a program with a set number of blocks. This task introduces learners to the programming construction of iteration (repetition).

Shortest Program

The program shown in **Figure 3** drives a triangular robot.

Following these instructions, the robot drives in a small square twice and ends up back where it started. (You can try it out if you wish, following the code in **Figure 4**.) The whole program uses five code blocks (three purple, one green, one grey).

Task

Write a program that gets the robot to the green square using twelve blocks or less.

This task is about spotting patterns and using loops to solve a problem.

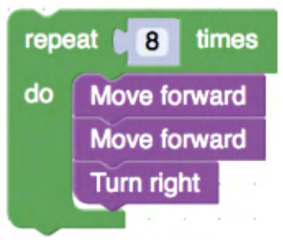


Figure 3 Program code for Shortest Program task

Pattern recognition (spotting patterns) in computer science and programming is key to determining appropriate solutions to problems and knowing how to solve certain types of problems. Recognising a pattern helps break down the problem and also build a construct as a path for the solution.

Expanding your toolkit

The previous three examples indicate how Bebras tasks can be used to either introduce, refresh, or reinforce programming concepts with learners. The quiz facility on the online UK Bebras platform allows teachers to create a quiz on a topic by selecting related Bebras tasks implemented on the platform and then share this online quiz for your learners to attempt. We hope that you might consider using a programming-related Bebras task as part of your teaching coding toolkit.

How to get your school involved

If your school is neither a UK-based school nor teaches a UK-based curriculum, then visit **bebras.org/countries** to discover if the Bebras Challenge is organised in your country and who to contact about having your learners participate in this year's Bebras Challenge.

If you are a UK-based education establishment or teach a UK-based curriculum, then visit **bebras.uk** for more information regarding the UK Bebras Challenge and how to register your school. This is so that your learners can participate in this year's UK Bebras Challenge. Once you've registered, you'll get access to the entire UK Bebras back catalogue of Bebras tasks, allowing you to create custom quizzes for your students to tackle at any time throughout the year. These quizzes are self-marking, and you can download your students' results to keep track of their progress. Schools have found these questions perfect for enrichment activities, end-of-term quizzes, lesson starters, and even full lessons to develop computational thinking skills. Join for free at **bebras.uk/admin**. (HW)

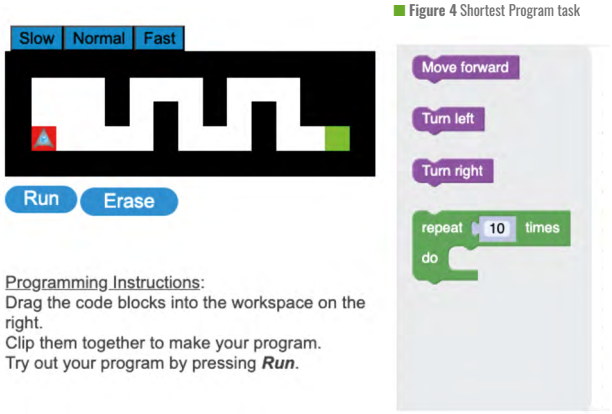


Figure 4 Shortest Program task

BEBRAS CHALLENGE 2025

This year's challenge will take place 10–21 November 2025.

In the UK, the challenge is open to all young people aged 6 to 19. Each participant has 45 minutes to tackle a series of interactive tasks, designed to encourage logical thinking and problem-solving skills appropriate for their age group. The challenge is conducted online, and tasks are marked automatically by our competition system. The tasks are designed to allow every student the opportunity to showcase their potential, whether they excel in maths or computing, or not.

MOONHACK 2025: A GLOBAL BIRTHDAY CHALLENGE

Blast off with Moonhack this year, celebrating ten years of coding challenges for young people

Kaye North

Moonhack (moonhack.com) is a free international event run by Code Club Australia (codeclubau.org) that brings kids aged 8–15 together from across the world for two weeks of coding fun. Over the last nine years, Moonhack has seen over 275,000 children coding Moonhack projects, with each new annual challenge presenting a unique theme about the world or space.

Moonhack's 10th birthday!

Moonhack began in 2016 as an initiative by Code Club Australia to get as many kids as possible coding on the same day.

What started as a local experiment quickly captured global attention — and in its first year, Moonhack set a world record with over 10,000 young people coding simultaneously. The achievement was a powerful statement: coding is not only an accessible endeavour, but also an exciting and unifying one for kids everywhere.

In 2017, Moonhack claimed another world record — this time, for the number of kids coding in one day, recording 28,575 children from 56 countries and 600 Code Clubs. In the early days of Moonhack, there were one or two projects for the day-long challenge. This has grown to six new projects for the challenge, with the introduction of micro:bits and design briefs. Moonhack also now takes place over two weeks, ensuring everyone can take part in the fun. Will you be joining in this year?

Featured projects

In 2025, for Moonhack's 10th birthday, there are six new space-themed birthday coding projects on offer. There is a coding project to suit everyone taking part (**moonhack.com/projects**):

- **Birthday blast off!** is a Scratch project for beginners. Featuring the voices of young people from Code Clubs around the world, this project is an animation that has a rocket countdown in different languages to blast off to a cake-shaped planet.
- **Lunar laughs** is a Scratch project suitable for both beginners and experienced users. The project has three steps. Step 1 is for beginners and includes setting up a birthday character to tell five space-themed jokes. Coders can finish their project there, or move on to Step 2 which introduces animation for the jokes. This could be the end of the project, or coders can continue to Step 3, which introduces lists and variables to create a random order for the character to tell the jokes.
- **Puzzle party** is a Scratch project suitable for experienced programmers. It asks participants to code a jigsaw puzzle by using blocks such as variables and absolute value, and by creating new blocks.
- **Partybit:mix** is a project that turns a micro:bit into a DJ mix board. Young people will program the micro:bit to store a list of music and a list of sound effects that can be combined to create a birthday playlist.
- **Astrolock** uses Python to create an escape room in a space shuttle: can coders escape the shuttle before it takes off? Coders can complete just one of the three steps or continue to the next, increasing the difficulty of the program. Step 1 asks participants to program a simple escape room, with one key and one way out. Step 2 involves the condition of the key being found in a random room. Step 3 includes programming to check multiple rooms and to add puzzles for each room.
- **Moonhack X** is this year's design project. Coders can choose any programming language and design a project in their own way that shares their Moonhack

Code Club members from Diamond Valley in Queensland, Australia, tested the Moonhack projects

> MOONHACK HELPS YOUNG PEOPLE TO BUILD VALUABLE CODING SKILLS AND INSPIRES THEM TO SEE HOW THEIR IDEAS AND CREATIVITY CAN MAKE A DIFFERENCE

experience. Coders could create a game about Moonhack, an animation about their journey with Moonhack, or anything else that they decide on!

Are you up for the challenge?

Getting involved in Moonhack is easy, free, and open to everyone — whether you're a teacher, Code Club leader, parent, or simply someone passionate about helping young people code. Join the birthday celebration this year between 14 and 30 October by visiting **moonhack.com** and registering your group, class, or club.

Once you have registered, decide on the projects that you will offer to your coders. All of the projects include step-by-step guides, translations, and classroom resources to make planning simple. There are also codealong videos for some of the projects, and blog posts to provide extra information and showcase the young people who tested the projects. You'll also find free downloadable resources such as posters, party bunting, unplugged activities, social cards, and certificates.

Participants can complete one or more projects during the challenge period — and once they're done, you can log their participation to count towards the global total. It's fun, flexible, and a fantastic way to celebrate coding!

Moonhack is your opportunity to be part of something truly global and exciting! By joining this challenge, you're not only helping young people build valuable skills in coding and problem-solving, you're also inspiring them to see how their ideas and creativity can make a real difference. With fantastic, free projects and a lively community of educators, families, and clubs worldwide, there has never been a better time to jump in. Let's celebrate Moonhack's 10th birthday together and code the future! (HW)

IMPORTANT DATES

Join the challenge live 14–30 October
Project submissions close 23 November

EFFECTIVE PROGRAMMING PEDAGOGIES IN THE INCLUSIVE CLASSROOM

How can programming pedagogies and teaching methods fully include learners with additional needs and disabilities in computing lessons?

There is a strong consensus in education that high-quality teaching is the most important factor in improving outcomes for all students, especially for those with additional learning needs and disabilities. Over the past few years, teachers and students have benefitted from an increased focus on research into what works when teaching programming in schools. If you haven't downloaded and digested *The Big Book of Computing Pedagogy* yet (helloworld.cc/bbcp), then it's probably time to do so!

In this article, I'll explore a selection of tried and tested approaches to teaching programming and how they support a range of learners, with reference to the twelve pedagogical principles developed by England's National Centre for Computing Education (**teachcomputing.org/pedagogy**).

Four areas of need

There are four recognised areas of need when considering young people with additional learning needs and disabilities:

- **Communication and interaction**: this includes autistic learners and children with speech, language, and communication needs
- **Cognition and learning**: this includes moderate and severe learning difficulties, and specific learning difficulties such as dyslexia
- **Social, emotional, and health difficulties**: this includes children with mental health difficulties and with attention deficit hyperactivity disorder
- **Sensory and/or physical needs**: this includes visually impaired or deaf learners, and children with a physical disability who require additional support

It is important to recognise that what works for one student won't necessarily work for another — even within the same area of need. A good knowledge of the young people you teach is essential to delivering high-quality teaching. The following approaches are just suggested starting points for supporting a wide range of learners.

Lead with concepts

There is a lot of technical vocabulary relating to programming, from 'algorithm' to 'variable'. Provide students with the opportunity to build a shared and consistent understanding of key concepts through the explicit teaching of vocabulary and the regular revision of terms. Image-supported glossaries can help with this, and you can find a programming-specific example for younger pupils here: **helloworld.cc/computing-glossaries**. Pre-teaching key terms will reduce demand on working memory, for example in learners with dyslexia and developmental language disorder. Deaf students will also be able to follow lessons more easily when technical terms have been introduced previously.

Autistic learners and young people with learning difficulties may struggle to comprehend abstract concepts. This is where unplugged activities can be useful, providing concrete contexts for

Catherine leads the new Computing & Digital Innovation Centre in Sheffield, UK (**helloworld.cc/learn-sheffield**), supporting schools with computing and the wider implementation of educational technology. She has over 20 years' experience of working with students with special educational needs and disabilities and their teachers, and is passionate about making computing accessible and inclusive for all learners.

GENERAL APPROACHES IN PROGRAMMING LESSONS

There are also a number of general teaching approaches that are essential to creating an inclusive programming classroom:

- Provide task sheets for students to refer to when completing longer tasks, to support executive function and keep them on task
- Establish classroom routines and provide structure for any group tasks to reduce anxiety
- Teach problem-solving approaches explicitly to support learner confidence
- Provide a list of relevant code snippets or physical code blocks in order to reduce cognitive load when planning out programs
- Plan programming projects that solve real-world problems to increase relevance to learners and raise engagement
- Include diverse role models and highlight career paths to inspire all young people

these terms; and harnessing movement, sound, tangible objects, and images will increase accessibility. Remember the semantic wave when you go unplugged, to ensure that learners can generalise what they have learnt about a concept (**helloworld.cc/semantic-waves**).

Get hands-on

Physical computing devices, such as floor robots, micro:bits, Crumbles, and Raspberry Pis, provide learners with a more tactile experience of learning to program. These devices have a range of sensory outputs, including movement, lights, and sounds, which can help learners with a visual impairment to fully access programming tasks. Making connections between the code and the physical output also supports learners with cognition and learning difficulties. Research has found that physical computing can be highly motivating for learners, and therefore it is a great approach for young people with social, emotional, and health needs who may struggle to engage in lessons. In a recent survey of teachers, physical computing was ranked highest from a number of approaches as being the most effective for teaching their autistic learners (**helloworld.cc/computing-pedagogy-autistic-students**), helping to make abstract concepts more tangible and harnessing meaningful, real-world contexts.

The Micro:bit Educational Foundation recently released an update to allow keyboard control in the MakeCode block-based programming environment, which is excellent news for learners with physical disabilities who prefer to use a keyboard rather than a mouse for access. They have also published guides for teachers working with students with sensory and physical difficulties (**helloworld.cc/makecode-accessibility**).

> “ THESE APPROACHES ENSURE EVERY LEARNER FEELS SUPPORTED

Structure lessons and foster program comprehension

Providing a range of ways for students to engage with code helps to ensure that all learners are included in programming lessons at a suitable level, and is a central tenet of the Universal Design for Learning framework (**helloworld.cc/universal-design-learning**).

I have written about PRIMM (Predict–Run–Investigate–Modify–Make) before, and this is a fantastic framework for providing different pathways through a lesson (**helloworld.cc/qr-PRIMM**). The majority of learners will be able to Predict the outcome of given code, and then Run it to test their prediction, allowing all young people to take part in lessons. The Investigate phase, where students annotate and explain the workings of code and complete debugging tasks, can be used to challenge learners and play to the strengths of, for example, autistic students, who may be good at detail-oriented tasks, without the executive function demands of creating a project from first principles. The Modify stage is great for scaffolding learning and building confidence by allowing a level of guaranteed success, and this can be extended by using Parson's Problems, where learners are asked to put code blocks or snippets into the correct order. PRIMM activities are also useful for creating short, focused tasks that keep the pace of a lesson high. This allows students, for example those with attention deficit hyperactivity disorder, to retain focus.

Computing and programming can provide opportunities for young people with additional learning needs and disabilities to excel. By applying the approaches outlined in this article, you can ensure every learner feels included, supported, and appropriately challenged. Explore the twelve pedagogies to identify strategies that break down barriers to learning, and amplify the unique strengths each learner brings to your classroom (**teachcomputing.org/pedagogy** or **helloworld.cc/bbcp**). (HW)

#INSIGHTS

BEYOND THE VIBE: HOW AI MIGHT UNDERMINE LEARNING — UNLESS WE DESIGN FOR FRICTION

STORY BY Manni Cheung

As a new researcher at the Raspberry Pi Foundation, I'm especially interested in how young people learn to program in an era shaped by artificial intelligence. Like many educators, I've been grappling with what it means to teach computing when powerful AI tools are already changing how code is written, explained, and understood.

During my master's degree, I conducted research that contributes to our developing understanding of teaching about and with AI. The central question I set out to explore was: What happens when students use AI to help them code, but end up learning less? I conducted a study with university students using tools like GitHub Copilot and ChatGPT to complete programming tasks. The results were both fascinating and cautionary, and highly relevant to computing education more broadly.

Students practised a style of coding now often called vibe coding: quickly improvising prompts, letting the AI tool write code, and only checking whether it "looked right". The approach was fast and frictionless, but often bypassed understanding. They could complete assignments, yet many struggled to explain how their code worked, or why it worked at all.

Although this research focused on undergraduates, the implications are increasingly relevant for K–12 educators. Generative AI tools are becoming more accessible, and younger students are beginning to encounter them in and out of the classroom. If we don't intervene with thoughtful guidance and intentional design, the habits of vibe coding could take root early, undermining foundational learning before it has a chance to develop.

Why friction matters

Coding is hard for a reason. Struggling to debug, finding the right syntax, breaking a problem into parts — these challenges aren't just obstacles, they're where learning actually happens. But AI tools are engineered to make these struggles disappear. The smoother the interaction, the easier it becomes for students to skip over the thinking.

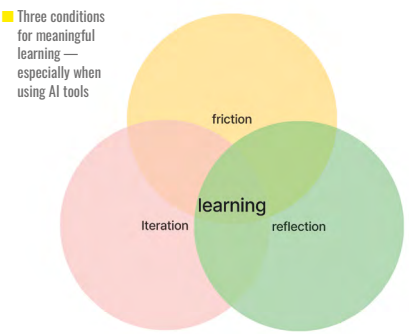
Yet here lies the paradox: the more AI is involved, the less actual interaction takes place. What was once a dynamic relationship between learner and machine becomes one of passive prompting and automated response. This raises concerns not only in education, but in the very fields of human–computer and human–AI interaction. If AI systems reduce learning to a transaction, rather than an exploration, we risk losing the reflective, dialogic qualities that make these technologies powerful partners in cognition.

What the research reveals

In my study, university students using AI coding tools were not blindly optimistic. They recognised the tension between getting quick results and actually understanding the work. Many described how AI encouraged skipping steps — prompting, accepting, and moving on. Several expressed concern about becoming overly dependent, and losing touch with debugging or problem decomposition.

What stood out, however, was their desire for more than seamless assistance. Some wanted the AI tool to pause and ask them to explain their reasoning first. Others suggested it could offer partial solutions or hints rather than full answers. One student put it simply: "I want to think first."

These preferences point to a deeper insight: students value challenge when it is framed as part of meaningful learning. They



don't want AI to think for them. They want it to support their thinking.

Bringing this into the classroom

For K–12 computing teachers, the question becomes: how do we introduce AI tools in ways that nurture reasoning, not just results? Here are a few practical strategies drawn from the research:

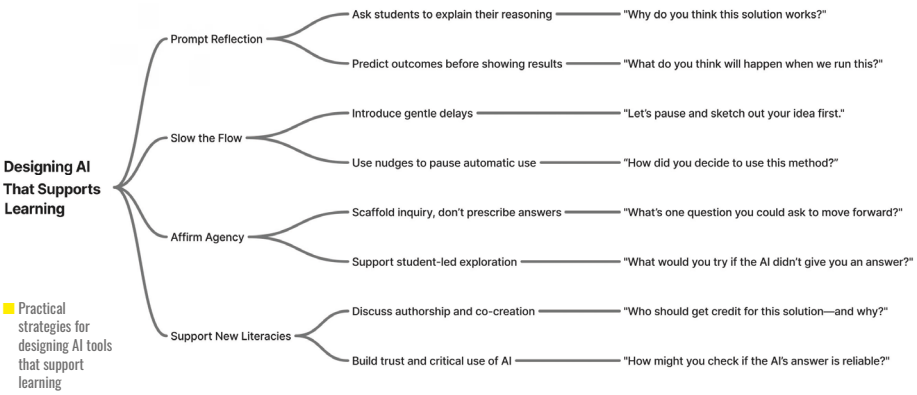
- **Introduce AI with structure:** frame tools like ChatGPT as part of a larger learning process, not just answer engines.
- **Create rules of engagement:** ask students to predict or explain what AI-generated code will do before running it.
- **Use AI to teach debugging:** prompt students to analyse flawed code, helping them build diagnostic skills.
- **Foster authorship conversations:** discuss what it means to co-create with AI. Whose ideas are being expressed?

These practices help ensure that AI remains a tool, not a shortcut that erodes understanding. They also encourage students to develop intentional habits of mind — skills that will matter far beyond the classroom.

From tool to companion

This reframing of AI as a learning partner rather than a solution engine opens space for thoughtful design. Based on the study, I propose three key strategies for building AI tools that support deeper engagement:

1. **Prompt reflection:** instead of auto-completing code, an AI tool might ask students to explain their logic or predict outcomes before confirming a solution.
2. **Slow the flow:** small delays, nudges, or even open-ended follow-up questions can disrupt automatic acceptance and reintroduce critical thinking.
3. **Affirm agency:** rather than over-directing, AI tools can scaffold inquiry by supporting student-led reasoning and reinforcing ownership of the process.



These strategies reposition AI not as an expert tutor, but as a curious peer — one that values thoughtfulness over speed, and inquiry over completion.

Supporting new literacies

The study also revealed that students are navigating broader questions about authorship, agency, and trust. Several likened their role in AI-assisted programming to that of a designer or curator — shaping prompts rather than writing code line by line. They expressed nuanced views about when AI use felt appropriate and when it compromised their learning.

This points to a broader pedagogical challenge. As generative AI becomes more prevalent, students need support not just in coding, but in cultivating critical digital literacies: when to rely on AI tools, how to question its suggestions, and how to maintain their own intellectual voice.

These are not technical skills alone. They are ethical and epistemic practices. And our educational tools should help learners navigate these choices with intention, not by accident.

This vision is echoed in one of our recent computing education research seminars, led by Professor Netta Iivari, titled 'Fostering transformative agency of children in the age of AI' (**helloworld.cc/iivari**). Drawing on examples from participatory and critical computing education, Iivari urged educators to create opportunities for children to not only use AI, but to question, critique, and reimagine it. Her work invites us to position children as active agents in shaping their digital futures, not passive recipients of algorithmic decisions. When AI is framed this way, it can empower young learners — not just accelerate their tasks.

Moving forward: keeping human thinking central in AI teaching

AI tools aren't going away, and students will use them — with or without our guidance. As computing educators, our task is not to resist this change but to shape it. We can introduce AI in ways that support autonomy, deepen reasoning, and preserve the slow, effortful parts of learning that make understanding possible.

As educator Randy Ziegenfuss observes (**helloworld.cc/ziegenfuss**), AI hasn't broken education — it has simply revealed what was already brittle. Many assessment models still reward output over understanding, and learning experiences too often prioritise speed and correctness over curiosity and process. The opportunity now is not to stop students from using AI, but to redesign learning so that it becomes so purposeful, so deeply human, that it resists automation altogether.

Vibe coding shows us what happens when fluency outpaces thought. But the deeper risk is systemic: the technologies designed to enhance human–computer interaction may strip away that interaction. In education, especially in early and formative years, we cannot afford to collapse the space between prompt and response. If we want students to learn, we must protect the struggle. And if we want AI to support that learning, we must teach it to ask, not just answer. (HW)

FURTHER READING

Iivari, N. (2025). *Fostering Transformative Agency of Children in the Age of AI.* Raspberry Pi Foundation Research Seminars, Cambridge, United Kingdom. **helloworld.cc/iivari**

Robbins, S. (2025). What machines shouldn't do. *AI & SOCIETY*, 40(5), 4093–4104. **helloworld.cc/robbins**

Ziegenfuss, R. (2025). *Generative AI Didn't Break Higher Ed — It Just Held Up a Mirror.* Retrieved from **helloworld.cc/ziegenfuss**

#INSIGHTS

FLARE: FRAMEWORK FOR LEARNING ABOUT RELATIONAL ELEMENTS

CODE COMPREHENSION THROUGH FAMILIAR STRATEGIES

STORY BY Justin Heath

"Programming was pretty straightforward when I was teaching Key Stage 1 [ages 5–7]. Now I'm moving to Year 5 [ages 9–10], it's ... well, I just hope someone else will teach it."

The language teaching connection

Computing and modern foreign languages are two of the three subjects that primary teachers are least confident with (**helloworld.cc/lives-of-teachers**), perhaps with good reason — both require guiding children through a potentially unfamiliar language. But here's the thing: primary teachers feel most confident teaching English, with well-tested, structured approaches for analysing, understanding, and creating text. What if we could apply some of those familiar strategies to programming, making code comprehension as routine as reading comprehension?

FLARE: revealing the structures of code

We've all seen it in our computing lessons: some children excitedly pile blocks onto the screen, keen to make something happen, but aren't sure why it works. Others hold back, overwhelmed by the unfamiliar look of code. While it's tempting to focus only on making things 'work', we must build understanding so that children develop the fluency to express their creative ideas.

FLARE (Framework for Learning About Relational Elements) is a method for leveraging common factors in literacy and programming. It adapts structured questioning from English lessons to block- and text-based programming. So I hope it is something you'll recognise, rather than see as something you need to learn.

The four elements of FLARE

Just as meaning in English is built from words, sentences, paragraphs, and purpose and organisation, the meaning of code is built at different 'scales'. FLARE breaks code into four elements; we can use them like lenses to think about different aspects of code (**Figure 1**).

How to use FLARE in the classroom

FLARE works with Scratch, MakeCode, 2Code, Python, and even C++ and Assembly Language. While FLARE has a 'blocks' element, it works equally well with text-based languages. Just read 'block' as 'statement' or 'command'. The same four elements reveal code structure regardless of syntax.

You don't need to throw out your curriculum. Try the FLARE lenses with one lesson or code snippet from your scheme of work. Leave out any parts that aren't relevant to your lessons.

Start simple this week

- Pick one lesson and add two FLARE elements: Block, and perhaps Relations. A Block prompt would be asking students to find individual blocks and explain what each does. A Relations prompt would be, 'When does this block run?'

GETTING STARTED: A CHECKLIST

- ✔ Choose one lesson and add a FLARE-style question
- ✔ Start with Blocks and Segments if you're new to code
- ✔ Make a question bank for different block types
- ✔ Team up with a confident programmer or computing lead if possible
- ✔ Allow yourself and your pupils a couple of weeks to get comfortable

You'll soon see deeper engagement and increased confidence in your lessons.

Figure 1 The FLARE elements make code structure visible and discussable, even for teachers who don't think of themselves as programmers

Element	What to look for	Purpose	Classroom examples
Block	Individual draggable code blocks with specific functions Literacy link: words clauses and sentences	Performs a single, basic action or instruction	■ Motion blocks (move, turn) ■ Looks blocks (say, change costume) ■ Control blocks (wait, repeat)
Segment	Connected blocks with a clear boundary, either a sequence running in order or blocks inside a control structure Literacy link: paragraphs	Defines a specific task involving multiple steps, repeats, or conditions	■ A sequence to move a sprite in a square ■ All blocks inside a **repeat 10** loop ■ The contents of an **if touching edge** block
Relations	Communication between different segments — the connections that aren't always visible Literacy link: connectives and cohesive devices	Coordinates how segments work together as a team	■ Event triggers (**when flag clicked** block) ■ Broadcasts between sprites ■ Variables affecting multiple segments ■ Function calls
Macro	Overall project design, organisation, and purpose Literacy link: purpose and organisation	Defines the program's main goal and how parts work together	■ 'This is a quiz game with three levels' ■ Flowchart of game logic ■ Storyboard for an animation

- Use real code from your scheme so the questions stick to what you're teaching.

Build your questioning routine over the next two weeks

- Make a quick question bank per common block type (events, motion, conditionals)
- Add one Segment prompt, 'What's this chunk trying to achieve?', and one Macro prompt, 'What was the programmer aiming for in this bit?'

Work with colleagues during this term

- Pair with a confident programmer/computing lead to co-plan a FLARE opener
- Swap a one-page slide that shows the four lenses and six to eight question stems you both like

Give it time (culture, not a trick)

- Expect one to two weeks for you and pupils to feel natural with the language
- Keep questions visible on the board or in a handout, so that pupils ask the questions unprompted

What teachers found

During a small-scale trial of FLARE in three classrooms, featuring teachers with a range of programming experience, clear benefits were seen:

- **Teacher confidence increased.** Even those with limited coding backgrounds said the clear structure "made it easier for me to understand personally, and then teach it more effectively".
- **Children became 'code detectives'.** They began predicting, explaining, and questioning code, rather than just following instructions.
- **Questions improved.** Teachers moved from generic 'What does this do?' to more targeted, differentiated questioning for both struggling and advanced learners.
- **Talk increased.** Lessons became more interactive, with children eager to share their ideas about how code worked.

> "FLARE ADAPTS STRUCTURED QUESTIONING FROM ENGLISH LESSONS TO BLOCK- AND TEXT-BASED PROGRAMMING

Challenges were also found:

- **Time.** Getting used to how FLARE works needed some time.
- **Relations and Macro.** These elements can feel abstract and confusing.

Why this approach works

FLARE builds on established educational theory: Piaget's progression from concrete to abstract (**helloworld.cc/piaget**), Vygotsky's social learning (**helloworld.cc/vygotsky**), and Bruner's spiral curriculum (**helloworld.cc/bruner**). Perhaps most importantly, FLARE recognises that meaning in code doesn't just live inside individual blocks or commands. Understanding comes from how parts relate — what triggers what, what follows what, and how chunks cooperate. FLARE helps you surface those relationships in everyday classroom talk.

I developed FLARE within the TICE2 project (**helloworld.cc/tice2**), and continue to refine it through classroom use and teacher feedback. You can find more information, including resources such as explainers and templates from the trials, at **jjgh.me**. (HW)

FURTHER READING:

Heath, J., Whyte, R., Sentance, S. (2025). FLARE: A framework supporting code comprehension and formative assessment in block-based programming education. *Computing Education Practice (CEP '25)* ACM. (**helloworld.cc/flare**)

Sentance, S., & Waite, J. (2021). Teachers' perspectives on talk in the programming classroom: language as a mediator. In A. Ko (ed.), *Proceedings of the 17th ACM Conference on International Computing Education Research*, 266–280. (**helloworld.cc/talk-in-programming**)

#INSIGHTS

SO WHY DO WE TEACH COMPUTING IN SCHOOL?

STORY BY Sue Sentance

The *Research, Insights* section of Hello World often includes summaries of research that relate to specific aspects of classroom practice. In the research I am describing here, the questions being asked are more fundamental. What do we mean when we talk about computing education in school? What is it? Why do we teach it? Who should be learning it? Now that computing education is embedded in many countries around the world (helloworld.cc/computing-education-worldwide), it feels like the subject is here to stay, so it's important to be able to have a shared language to enable us to discuss answers to these questions.

We do not all agree!

It seems these questions do not have universal answers. People may disagree in their answers to the 'why' and 'what' questions while still agreeing that computing is an important aspect of the school curriculum. In fact, this research was triggered by the fact that the instigators of the research project (me, Carsten Schulte, and Sören Sparmann) had noticed that computing education researchers were developing research initiatives in school and writing papers that seemed to suggest quite differing underpinning rationales or value systems. Thus, our key motivation was to understand and explain these differences so that people might be able to understand their own, as well as others', perspectives. Although we focused on researchers' views, findings are also relevant for classroom practitioners, those developing resources, and curriculum resources themselves.

Computing as a discipline vs. computing education

Back in 2008, Tedre and Sutinen proposed three perspectives on the discipline of computing, identifying that computing may be viewed as a mathematical subject, a scientific discipline, or may stem from engineering (**helloworld.cc/the-science-of-computing**). What this work did not introduce was computing as a societal discipline, which came out strongly in our recent research.

Thinking about computing education takes this debate one step further. If we understand why we teach a subject, it impacts the curriculum we develop. Consider the spider web diagram (**Figure 1**) proposed by the much-cited curriculum theorist Van Den Akker (**helloworld.cc/**

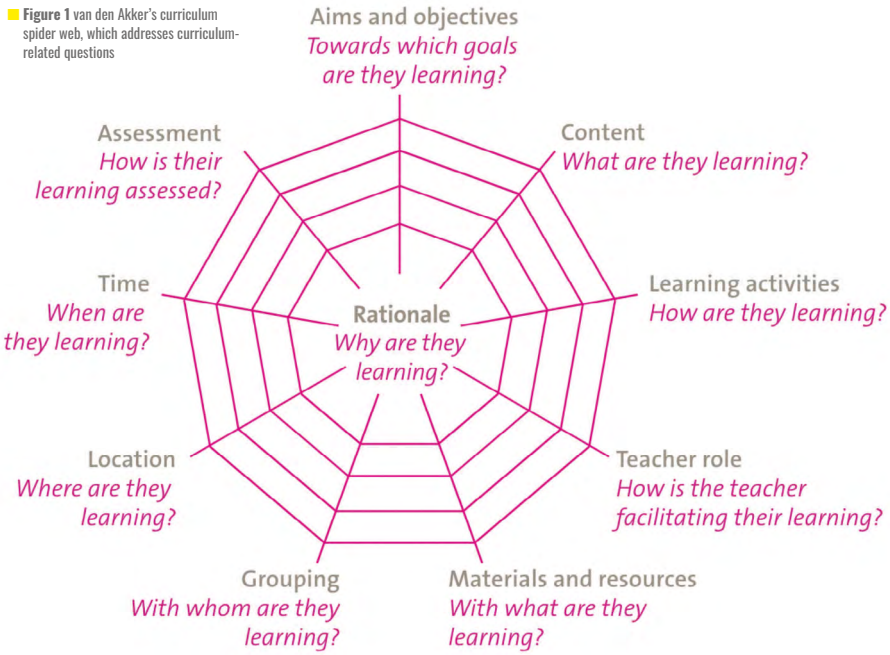


Figure 1 van den Akker's curriculum spider web, which addresses curriculum-related questions

Tradition	View of computing as a discipline	How it shapes computing education
Algorithmic/problem-solving	A formal practice that focuses on algorithms, computation, and transforming information	Encourages teaching abstract concepts and context-independent thinking
Scientific	A method of scientific inquiry aimed at understanding the world	Promotes modelling, simulation, and computational tools for inquiry
Design and making	An engineering science emphasising the design and construction of artefacts within constraints	Anchors projects in learners' creativity, interests, and collaboration to solve complex problems
Societal	A social practice shaped by its community, producing tools that embody and influence societal values	Inspires learners to examine the interplay of computing and society while fostering a sense of responsibility

Figure 2 Four traditions of computing education

> IF WE UNDERSTAND WHY WE TEACH COMPUTING, IT IMPACTS THE CURRICULUM WE DEVELOP AND IMPLEMENT

vandenakker). What this shows is that if we understand why students are learning, it impacts the content, learning activities, students involved, assessment approaches, etc. So understanding the rationale for teaching computing has a huge impact on how a curriculum (both national and local) is developed and implemented.

The results

A group of 15 researchers carried out this work. The approach used employed a mix of theoretical work and literature analysis. Considering the literature, two approaches were used. Firstly, a traditional review focusing on a set of around 60 papers which examined how the explicitly stated rationales of the authors impacted their work. Secondly, a larger scale review of over 1000 papers using Natural Language Processing and Large Language Model techniques to understand implicit rationales for research work. For the theoretical work, we considered philosophy and theory, beliefs about education, and perspectives on computing as a discipline (rather than computing education). Through much discussion and synthesis, we identified four main traditions of computing education (**Figure 2**).

These traditions highlight different aspects of computing and might influence curriculum design as well as research foci. The algorithmic/problem-solving tradition highlights the unique ways of thinking and the problem-solving inherent to computing, whereas the scientific tradition emphasises learning about the natural and artificial world through computational approaches. If your values point to the scientific tradition, you might teach computing showing how computing can be used across different domains. Both the design and making and the societal traditions focus on the real world: the former stresses the significance of implementing computational solutions within real-world artefacts, whereas if you teach computing based on the societal tradition, you might encourage students to reflect on the social context and moral implications of computing.

What is your perspective?

I believe we all have a perspective on what computing education is and what it is for, but that this might change over time. It will also change depending on our context (primary or secondary teaching level, vocational, etc.). We also may find that we can relate to ALL these traditions, but that one or two are much stronger for us. To examine this for yourself, try ranking the following four statements:

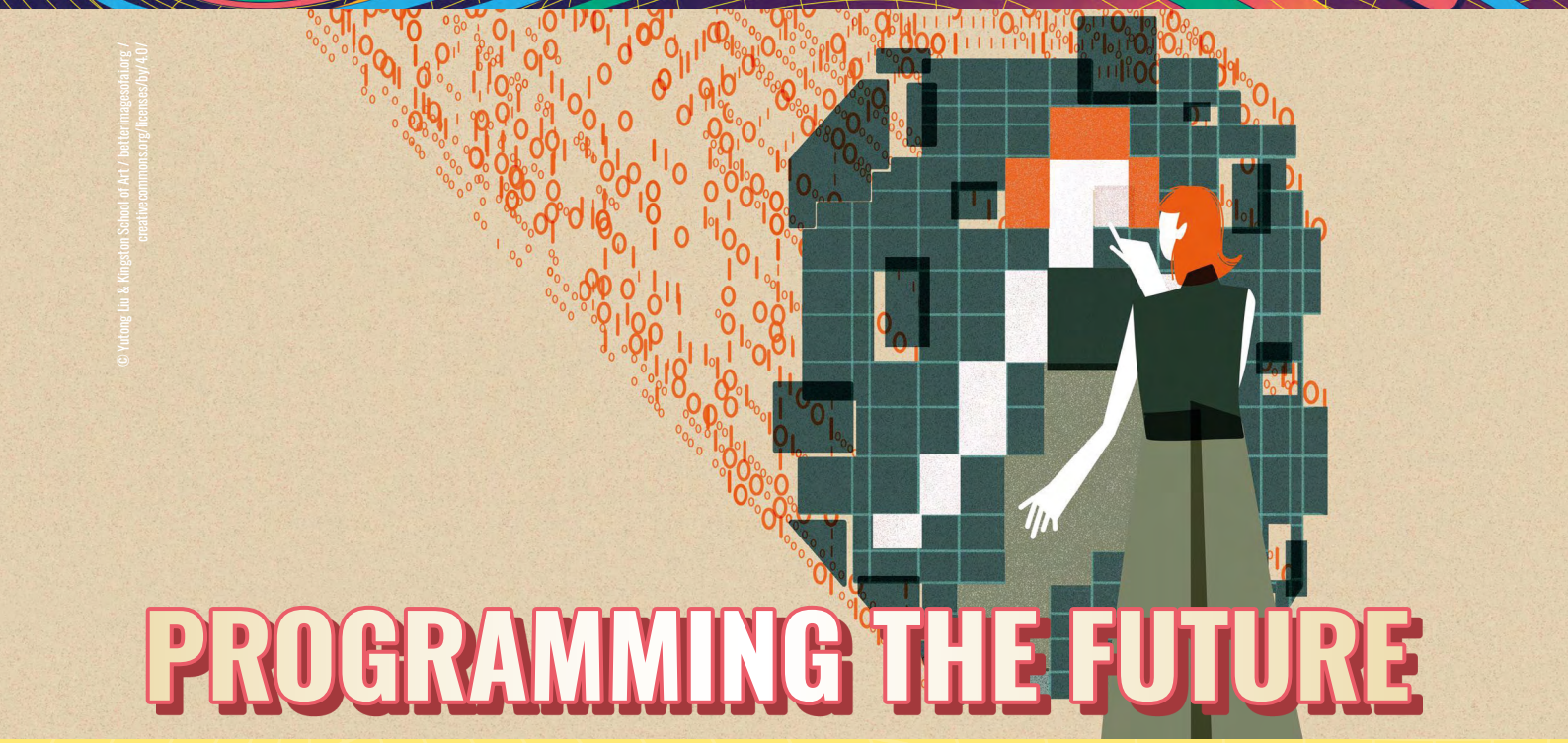
A) Building personally meaningful projects is the best way to learn computational concepts.
B) Students must learn how computing systems are shaped by societal values and power dynamics.
C) Understanding algorithms and computational processes is more important than understanding the data those algorithms manipulate.
D) Computing should be seen as a means to understand and explore the world, both natural and artificial.

Compare your ranking to others you work with to see if you share the same perspectives!

Thanks to all the researchers

Our working group included the following researchers: Carsten Schulte, Sue Sentance, Sören Sparmann, Rukiye Altin, Mor Friebroon-Yesharim, Martina Landman, Michael T. Rücker, Spruha Satavlekar, Angela Siegel, Matti Tedre, Laura Tubino, Henriikka Vartiainen, J. Ángel Velázquez-Iturbide, Jane Waite, and Zihan Wu, representing 13 different universities across 10 different countries. We worked on this from February to December 2024. Many thanks to them all!

The full paper is openly available online (**helloworld.cc/computing-education**) or there is a blog post with a little more detail on our website (**helloworld.cc/why-teach-computing**). We are now working on how this work translates to AI education. To keep up to date with the work of the Raspberry Pi Computing Education Research Centre, visit **computingeducationresearch.org**. (HW)



© Yutong Liu & Kingston School of Art / better imagesofai.org / creativecommons.org/licenses/by/4.0/

PROGRAMMING THE FUTURE

Simon Peyton Jones discusses programming as an act of creativity

"The programmer, like the poet, works only slightly removed from pure thought-stuff. He builds his castles in the air, from air, creating by exertion of the imagination. Few media of creation are so flexible, so easy to polish and rework, so readily capable of realizing grand conceptual structures." (Fred Brooks, *The Mythical Man-Month*, 1975)

Programming as an act of creation

What pictures or emotions does 'programming' conjure up for you?

For many, programming brings up negative images of socially challenged geeks staring at glowing screens in windowless basements. Or, somewhat more positively, an economically useful skill, but perhaps not a rewarding one.

But I think Fred Brooks has it right in the quotation at the start of this article. When you look at a jumbo jet, you get a visceral sense of its size, its power, its complexity, and the remarkable human creativity and achievement that makes it fly at all. But when you look at a computer running Linux, you just don't see the astonishing stack of clever, elegant, powerful layers of software that somehow conspire to put up that simple-looking window on the screen. It is the result of thousands of person-years of creativity and solid hard work — yet it is intangible, almost invisible.

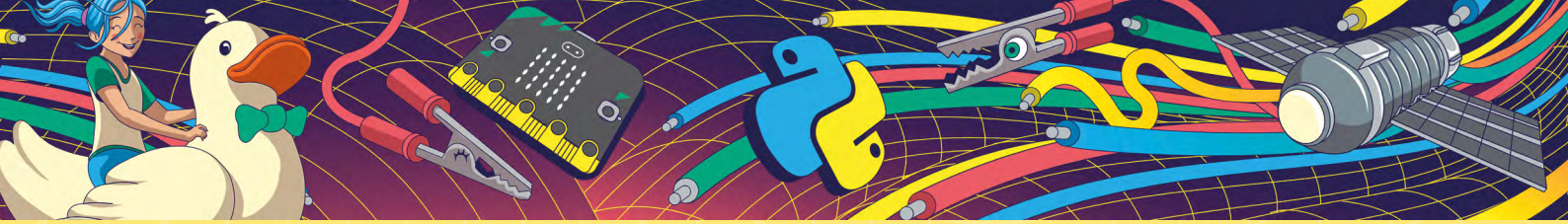
Designing and writing software is one of the most demanding, intellectually stretching tasks that humans undertake. If we build a building, we are limited by the strength of steel — we can only make the tower so high before it will fall under its own weight. But software knows no such limits. The only limit is our own ability (or inability) to manage the complexity of the systems we build. That is humbling — but also exciting.

So programming is, first and foremost, a creative activity. When you write a program, you bring into being something that has never existed before. It can be a thing of beauty or a complete mess — but it is yours.

What is programming anyway?

When we think of programming, we often think of writing a program to do something, starting from scratch. That's pretty hard! But when I think of programming, I think of a broad spectrum of other things, including:

- Understanding an existing program and predicting how it will behave
- Explaining to someone else how it works
- Modifying an existing program to do something new
- Starting from a program that is misbehaving: forming a hypothesis about what is wrong; designing experiments to test that hypothesis; performing those experiments; designing a fix to cure what is wrong — this is the scientific method in action
- Understanding a problem you want to solve and figuring out precisely what you want the program to do to solve it
- Using existing libraries and frameworks to build something wonderful — your program might be quite short, but it can leverage person-years of others' work
- Writing tests that explore whether the program meets its design goals, including writing the tests before you have written the program — this is often called 'test-driven development'
- Collaborating with others on a shared project, using platforms like GitHub



All of this requires careful, logical thinking, which is a very useful life skill in its own right. Computers are non-judgemental, but pitiless, judges of the precision of our thought.

But software development is not just nerdy logic-chopping. It is a rich, social activity, involving lots of interaction with other people, as we iteratively refine our shared vision of what the program should do and how we can design modular pieces to achieve that goal. It is very far from a sterile, mechanical process of 'coding'.

Will programming be rendered obsolete by AI?

Consider the question: how can I get this computer to do task X for me?

For most of computing history, the answer has been: 'Write a step-by-step program so that, when the computer blindly follows those steps, it does task X.'

The eye-opening thing about machine learning (the engine underlying AI) is that it suggests an entirely different way of telling a computer what you want it to do. Instead of a step-by-step program, show it a lot of examples of what you want it to do, and allow it to 'learn' what to do. That's an utterly different approach, and one that is data-centric rather than algorithm-centric. It is not magic — the mysterious 'learning' is actually just a lot of multiplications and additions — but it is very, very different. You might enjoy my talk 'Bits with soul' (2025), which explains how machine learning actually works (**helloworld.cc/darwin-codes**).

For some tasks, machine learning is definitely better: recognising cats, or playing Go, for example (**helloworld.cc/deepmind-go**). For other tasks, 'traditional' algorithmic programming is better: flying an aeroplane, for example. We should use critical judgement to choose the best tool for the job.

As programming copilots have shown, a generative AI system can even write many 'traditional' programs. Will programming (and programmers) therefore be rendered obsolete by AI? No, definitely not. Using AI tools to help write programs is like giving a carpenter a power drill: they can do more in the same amount of time; they can concentrate on rewarding activities, rather than repetitively drilling holes; they can be more productive; and they can have more fun.

The nature of programming may change, perhaps quite a bit. If you want to use a graphics library to draw a complex bar chart, that library will have dozens and dozens of functions, each with lots of parameters. AI copilots are really good at this stuff, so you don't have to remember the specifics of all those functions and their parameters. That can free you to spend time thinking about what sort of chart you wanted to draw in the first place, where the data comes from, and so on.

But copilots need pilots. A generative AI engine is just a machine that emits output that is, in a probabilistic sense, a likely response to a given prompt. That is far from a guarantee that it is the right output, and how could it be? Real programs have very complicated specifications, which are usually incomplete, seldom formalised, and constantly changing. An informally stated prompt, even a long one, is far from a full specification; or, to put it another way, many differently behaved programs might be reasonable outputs for the same prompt.

For all but the most throwaway code, a human (the pilot) must review, understand, modify, and sign off on the code that is produced. All serious software undergoes this kind of code review, even when it is written by human beings in the first place. And of course, those code reviewers must themselves be expert programmers.

In short, AI is not going to replace programmers: it can just make us more productive. Caution is in order though: AI tools make developers slower, but they think they are faster (**helloworld.cc/slow-down**).

You might enjoy my blog post from 2023 on this topic (**helloworld.cc/teach-code**), or the Raspberry Pi Foundation's 2025 white paper (**helloworld.cc/kids-code-2025**).

Programming the future

Douglas Rushkoff's book from 2010, *Program or Be Programmed* (**helloworld.cc/rushkoff**), asks the question: do we direct technology, or do we let ourselves be directed by it and by those who have mastered it?

The present is permeated by software, and the future even more so. We can choose to have agency over this ubiquitous software, or to be governed by it. 'Programming' may not mean exactly the same thing in the future as it has in the past, but it is the key to possessing that agency. I am proud to be a programmer. As William Ernest Henley puts it in his poem *Invictus*, "I am the master of my fate: I am the captain of my soul." (HW)

SIMON PEYTON JONES

Simon is a British computer scientist known for his work on functional programming languages, including being a key designer of the Haskell language and the Glasgow Haskell Compiler (GHC). He has held academic positions at University College London and Glasgow University, worked at Microsoft Research, and is currently an Engineering Fellow at Epic Games (**epicgames.com**). He is also a founder and chair of Computing at School (**computingatschool.org.uk**).

Guiding teachers in Kisumu County in western Kenya through their creative coding journey

UNLEASHING CREATIVITY

A new instructor's joy in learning to program in Scratch

My name is Esther Mmbai Diera, and while my foundation lies in the realms of mathematics and chemistry, my educational journey has taken an exhilarating turn, igniting a passion I never fully anticipated.

For years, I thrived as a classroom teacher, shaping young minds within the vibrant learning environment of Kakamega County in western Kenya. However, my path has converged with the dynamic world of educational technology at the African Maths Initiative (AMI; **africanmathsinitiative.net**). Here, I've not only honed my skills as a mathematics instructor but also discovered a thrilling new frontier: empowering educators through creative coding with Scratch.

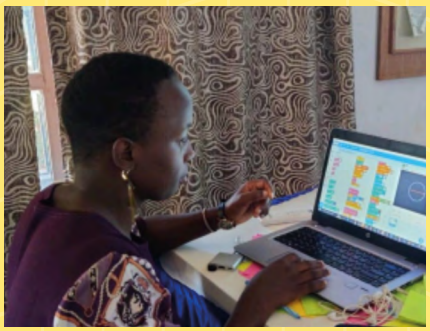
From blocks to breakthroughs

My introduction to this captivating software, orchestrated by AMI's visionary lead educator Mr Sam Okoth, came when I took a self-paced Raspberry Pi Foundation course.

It felt as if I was unlocking a hidden door to boundless imagination. I was instantly captivated by Scratch's playful, interactive nature. What unfolded over a mere two days was not just mastery of a software, but an awakening of my own creative potential, fuelled by the engaging learning paths that transformed education into an adventure.

Just a week later, the opportunity to co-facilitate my first creative coding workshop in Kisumu County arrived — a moment that was brimming with both excitement and a healthy dose of trepidation. The thought of sharing this newfound passion with my esteemed colleagues stirred a nervous energy within me. Yet, the bedrock of my classroom management skills and my comfort in public speaking proved to be invaluable anchors.

Supported by my incredible team, those initial moments of uncertainty soon melted away and were replaced by a confident stride. As I guided my peers through the wonders of Scratch, the experience was transformative, not just for the teachers I was instructing, but for myself as well. The teachers' unbridled creativity and infectious enthusiasm became a powerful source of learning and inspiration.



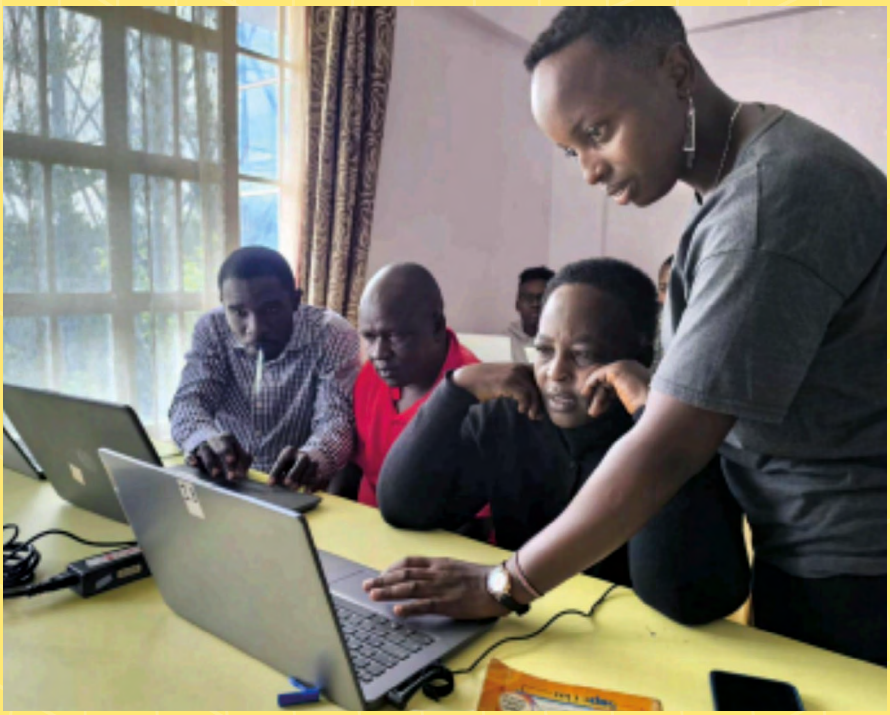
Exploring the Scratch interface and preparing to guide teachers on this exciting coding adventure

Frameworks for facilitating

Two months later, I facilitated sessions at the Kongoni Creative Coding Workshop (**kongoninetwork.org**) back in Kakamega County in western Kenya. With its theme of 'Empowering Educators with Scratch for Creative Learning', the workshop further solidified my commitment to this transformative tool. We immersed ourselves in hands-on exploration within the accessible Scratch environment, guided by Code Club's '3...2...1...Make!' framework (**helloworld.cc/3-2-1-guide**) together with the cyclical magic of Scratch founder Mitchel Resnick's 'Creative Learning Spiral' approach (**helloworld.cc/creative-learning-spiral**).

Witnessing educators beginning to embrace these concepts and develop their foundational coding skills was truly exhilarating. Facilitating three sessions and co-leading others alongside my dedicated colleagues felt remarkably natural. The ease with which I could now guide teachers through the platform, troubleshoot challenges, and witness their 'aha!' moments filled me with an immense satisfaction.

The follow-up training was perhaps the most rewarding experience yet. To witness the tangible progress the teachers had made in the intervening weeks, and the innovative ways in which they were already integrating Scratch into their classrooms, was nothing short of inspiring. Leading the initial session, in which they shared their unique journeys and tackled implementation challenges, highlighted their growing confidence and ingenuity. Their project presentations were a powerful testament to their mastery, a symphony of digital creativity that resonated deeply within me as their facilitator.



Guiding teachers in Kongoni as they navigate the Scratch environment and discover the power of coding to engage their students

> WITNESSING THE PROGRESS TEACHERS HAD MADE WAS NOTHING SHORT OF INSPIRING

Igniting future creators

Creative coding using Scratch has profoundly impacted me, both as a continuous learner and as an educator. It has unlocked a vibrant space within me where imagination and creativity can flourish, and coding has evolved from a skill into a cherished hobby. Now, my driving force is to ignite this same spark in as many educators as possible. I envision a future where learning is infused with the joy and engagement that Scratch offers. The prospect of introducing Scratch in Code Clubs fills me with excitement, knowing it will provide learners with a platform to code, create, and enjoy the process of discovery. My journey as a Scratch instructor is not just about teaching code; it's about empowering educators to unleash the creative potential within themselves and their students, paving the way for a more innovative future for all. (HW)

ESTHER MMBAI DIERA

Esther is a passionate chemistry and mathematics educator who is dedicated to sparking curiosity, confidence, and creativity in every learner (**LinkedIn: Esther Diera**).

WHAT PROGRAMMING LANGUAGE TRANSFER TEACHES US ABOUT HELPING STUDENTS TRANSITION

Ethel Tshukudu draws on her research into programming language transfer, helping educators guide students from blocks to text

As educators, we celebrate when students master their first programming language, often block-based (like Scratch) or procedural Python, and begin to express their ideas through logic, structure, and creativity. But what happens when those same students encounter their second language, typically a text-based one like Python or object-oriented Java?

ETHEL TSHUKUDU

Ethel teaches introductory Java programming at San José State University, USA. Her PhD research focused on programming language transfer, and she supports global computing education through CSEdBotswana and ACM SIGCSE Board service (**@EthelTshukudu, LinkedIn: Dr Ethel Tshukudu**).

My PhD research into programming language transfer offers useful insights. In that work, we investigated how students transition between text-based languages, particularly from procedural Python to object-oriented Java (**helloworld.cc/conceptual-transfer-programming**).

We found that students often carry over assumptions from their first language; some helpful, some not helpful. Whether these assumptions support or hinder learning depends on how cleanly the concepts transfer. The research further proposes a pedagogy of transfer to help teachers anticipate and support these transitions (**helloworld.cc/conceptual-transfer**).

Understanding programming language transfer

Drawing on natural language acquisition theories, we identified three types of conceptual transfer students experience when learning a second programming language during code comprehension:

1. True Carryover Concepts (TCCs)
These are constructs that share similar syntax and semantics across languages, enabling successful transfer.

Example:
Python:

```
if x > 5:
   print("x is greater than 5")
```

Java:

```
if (x > 5) {
   System.out.println("x is greater
   than 5");
}
```

Because the structure and meaning are nearly identical, students usually understand and apply them correctly and implicitly. Teachers can move quickly through these.

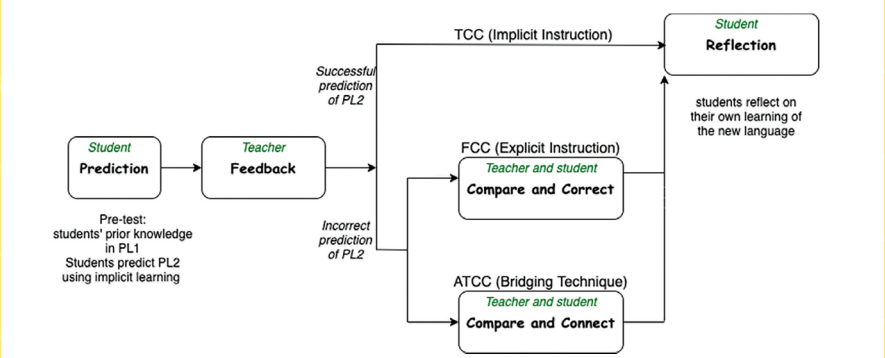
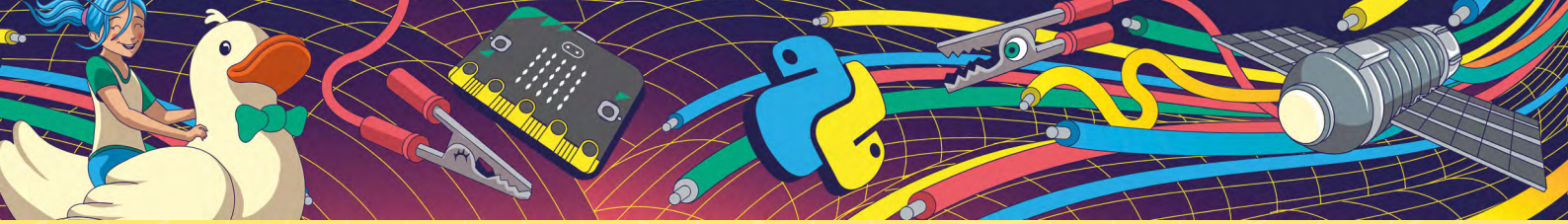


Figure 1 Transfer Pedagogy Framework, which helps educators to recognise the type of transfer involved and to respond with targeted strategies



2. False Carryover Concepts (FCCs)
These constructs have similar-looking syntax across languages but behave differently, often leading to misconceptions.

Example:
Python:
```
e = [1, 2, 3]
f = [1, 2, 3]
print(e == f)  # True (compares
contents of the list)
```

Java:
```
int[] e = {1, 2, 3};
int[] f = {1, 2, 3};
System.out.println(e == f);  //
False (compares references of
arrays)
```

Here, students incorrectly assume that equality behaves the same way in Python and Java. This is where misconceptions arise. It is also where we can compare and correct understanding (**helloworld.cc/conceptual-transfer**), and use the opportunity to introduce deeper concepts, such as memory references, that were abstracted away in the first language.

3. Abstract True Carryover Concepts (ATCCs)
These concepts share similar meaning but differ in form and representation between the languages.

Example:
Procedural Python:
```
dog = {"name": "Fido", "age": 5}
```

Java:
```
class Dog {
   String name;
   int age;
   Dog(String name, int age) {
        this.name = name;
        this.age = age;
   }
}
Dog dog = new Dog("Fido", 5);
```

While Python dictionaries and Java objects differ in syntax and structure, both represent structured data associated with named attributes, making them conceptually similar from a learner's perspective. Teachers can support transfer by bridging understanding through analogy, similar examples, and guided comparisons, as shown in the example.

The transfer pedagogy framework

To support effective teaching across languages, we proposed the Transfer Pedagogy Framework (**helloworld.cc/conceptual-transfer**), shown in **Figure 1**. The model begins with a student prediction of how a concept from Language 1 applies to Language 2. The teacher then provides feedback based on that prediction:

- If the prediction is correct, it is usually a TCC, and learning proceeds with minimal intervention
- If the prediction is incorrect, the teacher guides the student to compare and correct (FCC) or compare and connect (ATCC)

The process concludes with student reflection, allowing learners to consolidate and refine their understanding.

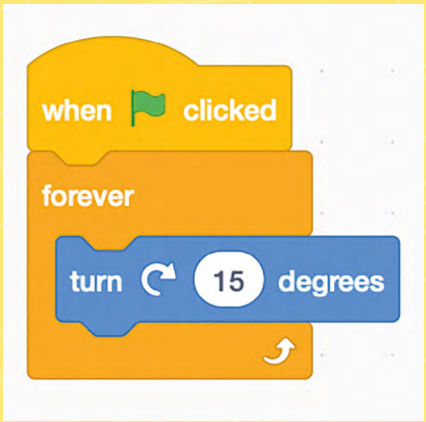
Applying the transfer research to block-to-text transitions

Earlier work by Weintrop and Wilensky shows that block-based environments reduce syntactic load, enabling early focus on logic and structure, which are key foundations for conceptual transfer (**helloworld.cc/comparing-programming**). This may explain why their research reported that students taught with blocks outperformed their text-based peers, even on text assessments, demonstrating near transfer of core concepts. While my research focused on text-to-text transitions, I believe my research is valuable when helping students transition from block-based tools like Scratch to text-based programming in Python. In the case of Scratch to text, we may be encountering a lot of ATCC concepts, where there are lots of differences but concepts are the same.

STRATEGIES FOR SUPPORTING TRANSFER LED BY THE TEACHER

Strategy	Description	Transfer Type
Match familiar code	Use familiar constructs shared across languages, such as 'if' statements, and show their equivalent forms in Python, as students typically grasp these quickly and require minimal reteaching.	TCC
Contrast behaviour	Use similar-looking code that behaves differently across languages to prompt discussion, correct misconceptions, and introduce deeper concepts.	FCC
Bridge abstract concepts	Use analogies or visuals to connect concepts, for example, sprites to objects and blocks to functions when moving from Scratch to Python. Reinforce the connection by using consistent variable names and access patterns across languages to highlight conceptual similarities. Where possible, allow learners to toggle between block and text modalities to support side-by-side comparison.	ATCC

For example, in Scratch:



This is conceptually simple: an event, a loop, and an action. But its Python equivalent in trinket (**trinket.io/python**):

```
import turtle

screen = turtle.Screen()

cat = turtle.Turtle()

while True:
    cat.right(15)
```

The Python version introduces imports, object creation, and method calls, none of which are explicit in Scratch, where the sprite is treated as an object and actions are hidden method calls. As a result, students may fail to predict how these concepts transfer. In these cases, the teacher must step in to compare and connect representations, an ATCC scenario in the framework.

Why this matters for teaching

This transition isn't just about teaching syntax, it's about helping students manage the cognitive load that Scratch once handled for them: from setup and graphics to object-oriented structure and control flow. For many, the sudden complexity of text-based languages can be overwhelming.

Tools like Trinket, which support Python turtle graphics in the browser, can help bridge this gap. They preserve the visual, interactive feel of Scratch while gradually introducing Python syntax. This aligns with the 'compare and connect' strategy for ATCCs in the framework. However, teacher intervention is crucial during transfer.

Connecting to broader work

Other research shows that dual-mode environments, where learners can toggle between blocks and text, support a range of student strategies, such as comparing and copying, and hence encourage explicit comparisons across representations (**helloworld.cc/switch-mode**).

Similarly, Guzdial demonstrates the power of scaffolding in a cross-representational sequence: students begin with a simplified 'teaspoon' language, then move to Snap!, and finally move to equivalent Python code using Runestone Academy e-books (**helloworld.cc/scaffolded-programming-arts**). His design, inspired by conversations around programming language transfer, emphasises the importance of surface alignment and semantic continuity to promote conceptual transfer.

Final thoughts

How we teach the first programming language shapes how students learn the second. In block-based tools like Scratch, students focus on logic and structure. In text-based languages like Python, they must navigate syntax, abstraction, and complexity. Using the Transfer Pedagogy Framework, educators can anticipate where students will thrive, where they'll struggle, and how to guide them across paradigms intentionally.

In essence, teaching a second programming language is not just a technical shift, it's a pedagogical opportunity to do the following:

- Surface prior assumptions
- Leverage transferable knowledge
- Build deeper, more flexible mental models of programming

Whether students are moving from Scratch to Python, or from Python to Java, understanding programming language transfer helps us teach more intentionally, transforming what might seem like a hurdle into a powerful learning moment. (HW)

FURTHER READING

Guzdial, M. (2023, July 10). A Scaffolded Approach into Programming for Arts and Humanities Majors. *Computing Ed Research — Guzdial's Take.* (**helloworld.cc/scaffolded-programming-arts**)

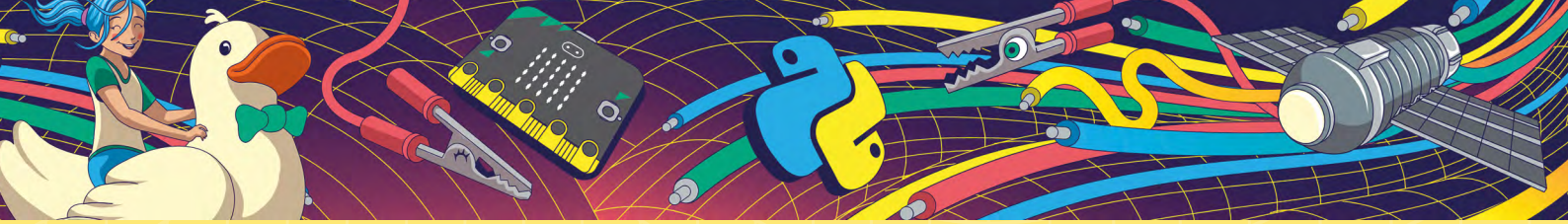
Lin, Y., Weintrop, D., & Mckenna, J. (2023, June). *Switch Mode: Building a middle ground between Block-based and Text-based programming.* In *Proceedings of the 2023 Symposium on Learning, Design and Technology* (pp. 114-118). (**helloworld.cc/switch-mode**)

Tshukudu, E., & Cutts, Q. (2020, August). Understanding conceptual transfer for students learning new programming languages. In *Proceedings of the 2020 ACM conference on international computing education research* (pp. 227-237). (**helloworld.cc/conceptual-transfer-programming**)

Tshukudu, E., Cutts, Q., & Foster, M. E. (2021, November). Evaluating a pedagogy for improving conceptual transfer and understanding in a second programming language learning context. In *Proceedings of the 21st Koli Calling International Conference on Computing Education Research* (pp. 1-10). (**helloworld.cc/conceptual-transfer**)

Weintrop, D., & Wilensky, U. (2017). Comparing block-based and text-based programming in high school computer science classrooms. *ACM Transactions on Computing Education (TOCE), 18*(1), 1-25. (**helloworld.cc/comparing-programming**)

> TEACHING A SECOND LANGUAGE IS A PEDAGOGICAL OPPORTUNITY



STRYPE: A FRAME-BASED APPROACH TO PYTHON

Simplifying Python for beginners with intuitive frame-based editing and interactive graphics

Novice programmers often start their learning journey with block-based editors, and later focus on traditional text-based editors. The transition between these two different editing paradigms is known to be difficult, as students may not easily transfer the knowledge they acquired while using blocks to the textual counterpart in a text-based editor, and in a programming language that differs in syntax.

To facilitate this transition, Strype, an online editor for Python, proposes an intermediate approach using frame-based editing. This approach combines the safeguards and simplicity of blocks, with the flexibility and realism of text. With educational purposes in mind, Strype can be used to create different kinds of applications.

What is Strype?

When encountering a new programming environment, a teacher typically asks: What is it? What is new or different? And why should I care?

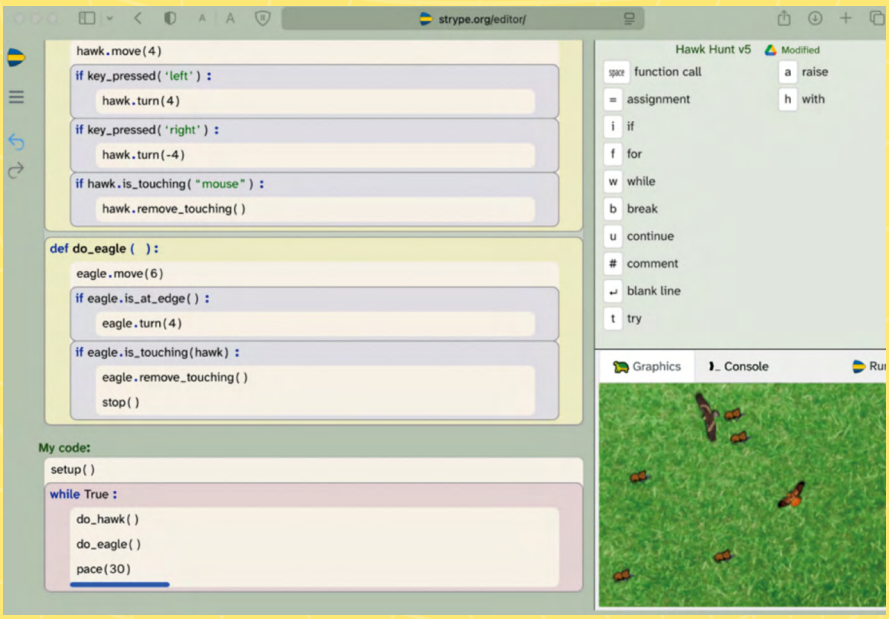
Let's start with what it is. Strype (**strype.org**) is a free, web-based Python programming environment designed for novice programmers. Strype operates entirely in a web browser, eliminating the need for installation and allowing users to start programming instantly. Its unique graphics capabilities allow for more engaging introductory examples than traditional "Hello World" programs, fostering motivation and engagement while still teaching fundamental concepts like variables, 'if' statements, and loops. Finally, its innovative frame-based editing blends elements from block-based and text-based editors to offer the best of both worlds: simplified statement-level code entry and error reduction, combined with the organisation and the power of a professional language.

What is different about Strype's programming environment?

Unlike traditional text editors where users type character by character, Strype's frame-based editor allows an entire statement, such as an 'if' statement or 'for' loop, to be entered in a single interaction. Upon opening Strype, a minimal Python program appears with a blue bar, the 'frame cursor', indicating where new frames can be inserted. This cursor can be moved up and down using arrow keys.

Frames: every statement in Strype is represented as a 'frame' inserted into the program using a single command key. A complete list of available frames and their command keys is displayed in the top right-hand side of the Strype interface, adapting contextually to the cursor's position. For example, pressing 'i' inserts an 'if' statement frame. Once inserted, frames contain 'slots' to fill in more code.

Slots: frames have two types of slots: 'frame slots' which accept other frames, and 'text slots' which accept text. In an if-frame,



The Strype system runs in a web browser, with graphics capabilities and frame-based editing

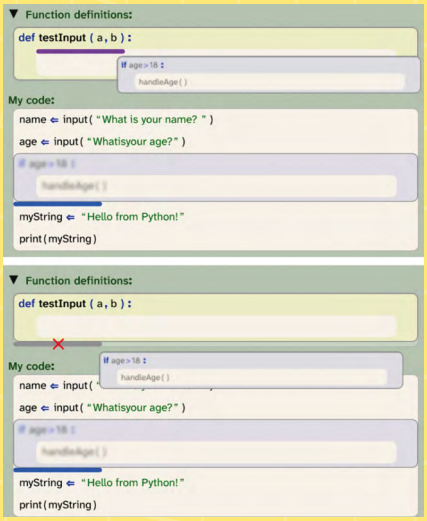
THE ADVANTAGE OF FRAMES

Frame-based editing offers several advantages over both text-based and block-based editing:

- **Reduced syntax memorisation:** novices don't need to remember complex syntax rules (such as keywords like 'in' or colons for loops); the editor handles this automatically, allowing learners to focus on semantics.
- **Reduced typing effort:** for younger learners, typing can be a significant cognitive load. Frames reduce this by creating entire statements with a single keypress, leaving only text slots to be filled.
- **Improved readability and layout:** coloured frame backgrounds enhance the visual clarity of program structure, and automatic layout prevents indentation errors.
- **Enhanced discoverability:** the frame menu provides a constant list of available statements, reducing the need for memorisation.

Compared to block-based systems, frame-based editors offer:

- **Keyboard-driven efficiency:** while mouse gestures are supported, all editing can be done with the keyboard, improving efficiency for proficient users.
- **Reduced 'viscosity':** editing at the expression level is primarily keyboard-based, avoiding the frustration of dragging new blocks for every small element (like '+') common in block-based systems.
- **Traditional program layout:** Strype arranges program statements linearly, similar to professional programming systems, making code more understandable and maintainable than the scattered, two-dimensional layout often found in block-based environments. This facilitates easier code reading, debugging, and management of larger programs.



Strype allows programmers to drag and drop frames but only with a valid drop target (top of image) vs. an invalid drop target (bottom of image)

the condition is a text slot, and the body is a frame slot. When editing, the system indicates the slot type with either a text cursor (for text slots) or a frame cursor (for frame slots). Slots with a white background are mandatory and must be filled, while cream-coloured slots can be left empty.

Moving frames: frames can be moved using the mouse (drag and drop) or by selecting them with Shift keys for cut, copy, and paste. Strype's editor enforces syntactic validity, only allowing frames to be dropped or pasted at correct positions. This ensures the program always maintains correct structure. Right-clicking a frame reveals a context menu which displays more editing options.

Layout: Strype automatically manages code layout, including indentation, whitespace, and line wrapping. This is particularly beneficial for novices, especially in Python where indentation defines code structure, as it eliminates common indentation errors.

Why should I try Strype?

Strype includes various editing supports that improve on text-based editors, making programming easier, quicker, and less error-prone.

Flexible control structures: control structures like 'if' statements or loops can be flexibly added or deleted. If frames are selected, inserting an if-frame will surround the selection, using it as the body of the 'if' statement. When deleting a control structure, users can choose to delete the entire statement (including its body) or just the outer frame, leaving the body in place, all with a single keystroke or via the context menu.

Parameter prompts: when entering a function name in a function call frame, prompts for required parameters are automatically displayed. This helps programmers remember parameter names, purposes, and with indicating missing arguments.

Code completion: Strype supports code completion, similar to other integrated development environments. It is triggered manually by typing Ctrl+Space (rather than automatically) when entering a function call, type name, or identifier. This prevents distractions for beginners and allows users to choose when to use the feature.

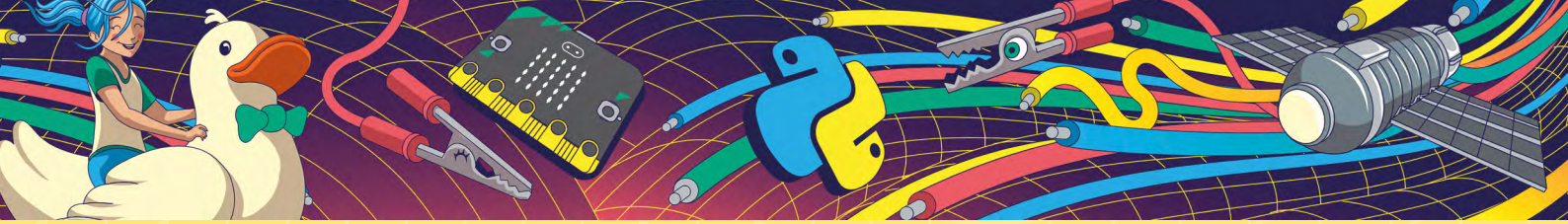
Programming with graphics: Strype supports standard Python programs with text-based console input/output and includes a text console and standard Python turtle graphics library. However, its novel interactive graphics support allows for more visual and engaging examples from the start.

Bring programs to life with graphics

Strype provides a graphics library, imported in the 'Imports' section of a program, which enables graphical output. When a program produces graphics, the output area automatically switches from the text console to the Strype graphics world.

Media literals: images can be used as literal values. To set the background of the graphics world, the 'set_background' function can be used. Users can copy an image (from the internet or local files), place the cursor in the parameter slot, and paste it. A small thumbnail of the image is displayed in the code. Hovering over the image literal provides simple editing options like scaling or cropping. Sound literals are also supported and can be similarly copied and manipulated. The graphics library also includes an actor class, which can receive an image when created, and actor objects are automatically displayed in the graphics world.

Animation: once an actor is on screen, it can be moved using functions available in the graphics API. Animation programs typically involve an 'infinite' loop. The pace function



controls the execution speed of the loop, delaying execution to refresh the screen at a specified rate (e.g. 30 frames per second).

Interaction: To create games, additional graphical actors and interaction are needed. Strype supports reacting to keypresses. For instance, pressing left/right cursor keys can turn an actor, creating a keyboard-controlled animated character with minimal code. More complex game examples are available in the Strype menu under 'Examples'.

> STRYPE COMBINES THE SAFEGUARDS OF BLOCKS WITH THE REALISM OF TEXT

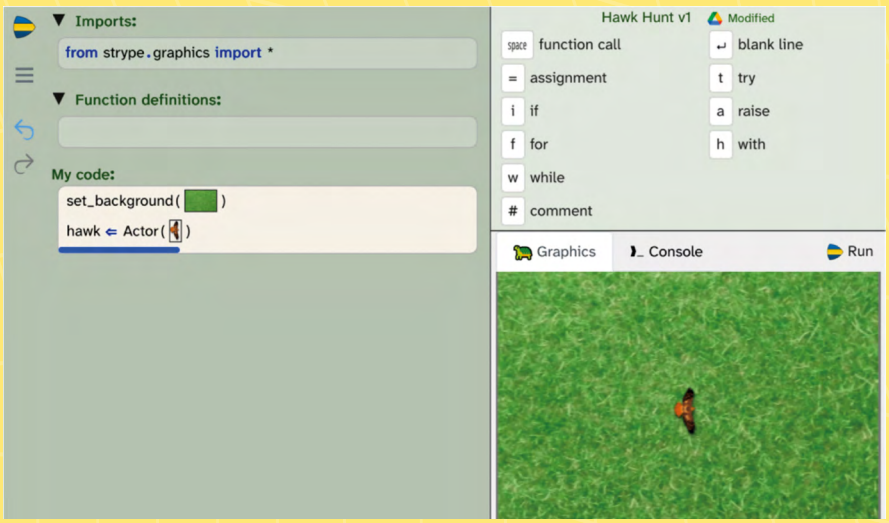
Environment support

The Strype web-based environment offers several features for teaching programming:

- **Saving programs:** programs can be saved locally or to Google Drive (with future support for other cloud storage systems).
- **Compatibility with standard Python**: standard Python code can be pasted into the Strype editor and automatically converted to Strype frames. Existing Python programs can also be loaded, simplifying the use of existing materials.
- **Image conversion**: individual frames or sequences of frames can be converted into images via a built-in function from the context menu, facilitating the creation of worksheets and teaching materials.
- **Program sharing**: programs saved in cloud storage can be easily shared with others, such as students, via a link. Clicking the link loads a copy of the shared program, which users can then work with and save independently.

There is much more to discover about programming in Strype. We cannot discuss all the details here, so we will end by pointing you to the most valuable resource when working with Strype: the Strype teachers' lounge. Accessible from **strype.org**, it is open to teachers at public teaching institutions interested in using or discussing Strype. The website is private to prevent pupils from listening in, allowing for free discussion of teaching material and strategies. The Strype development team is also present to discuss ideas for system improvements and extensions.

Our goal is to make programming more accessible to more learners, and we hope that Strype will be another tool in your toolkit to bring your students' programs to life. (HW)



A simple program with graphics output

MICHAEL KÖLLING

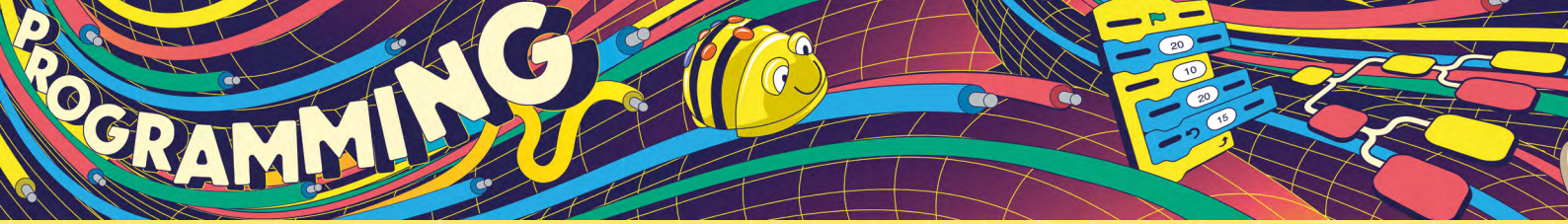
Michael is professor of computer science at King's College London. He is a researcher in computing education and author of programming textbooks. He works on educational development environments. His past projects include the BlueJ and Greenfoot systems.

NEIL BROWN

Neil is a senior research fellow King's College London and works as a researcher and developer on several educational programming environments, including BlueJ, Greenfoot, and Strype.

PIERRE WEILL-TESSIER

Pierre is a research associate at the King's College London Programming Educational Tools (K-PET) research group, which developed and maintains BlueJ, Greenfoot, and Strype.



FROM BASIC TO PYTHON: A HISTORY OF PROGRAMMING LANGUAGES

Diane Dowling traces the use of programming languages in UK CSed and explores the global impact of Python

When I started teaching computer science in the early 2000s, the UK did not have a national curriculum, and the only CS that was taught was a fairly niche A-level qualification for students aged 16–19. (Advanced-level qualifications are subject-based qualifications that can lead to university, further study, or work.)

The students who enrolled were often already accomplished programmers, having been self-taught or tutored by an enthusiastic parent. They were almost all boys, and most of them — outside school — lived in their bedrooms writing code. This is the cohort from which teenage hackers such as the infamous Gary McKinnon and Daniel Cuthbert emerged, figures whose curiosity with computers led them into high-profile trouble before some later transitioned into cybersecurity careers.

Visual Basic

Like many of my teacher peers, I used Visual Basic (VB) as a programming language to deliver the programming components of the computer science A level. Most UK textbooks provided code examples in VB, and the resources I inherited from the teacher I took over from were also VB-based.

VB had a distinguished pedigree. BASIC (Beginners' All-Purpose Symbolic Instruction Code) was promoted through the BBC Computer Literacy Project in the 1980s and continued to be used into the 1990s. Many UK schools retained BBC Micros well into the 1990s because of budget constraints and the robustness of the machines. In the mid-to-late 1990s, QBasic gained traction in schools that moved to IBM-compatible PCs.

As the 1990s progressed, and more schools upgraded to Windows-based PCs, VB emerged as a natural successor to earlier BASIC dialects. VB offered a graphical user interface (GUI) design environment, making it especially appealing in an educational context, where drag-and-drop interactivity supported visual learning and rapid application development. It allowed students to create simple programs with buttons, text boxes, and menus, bridging the gap between procedural code and event-driven programming. As schools moved into the 2000s, VB became the programming environment of choice.

The rise of Python

A major shift occurred in the 2010s, when the UK government overhauled the computing curriculum. In 2014, ICT was replaced with computing as a statutory subject, placing a renewed emphasis on programming, algorithms, and computational thinking from Key Stage 1 (ages 5–7) onward. This marked a move away from teaching how to use applications and towards teaching how to design and build them.

It was in this context that Python rose to prominence. Favoured for its simple, readable syntax and its wide use in industry, Python was rapidly adopted by teachers, exam boards, and education-focused organisations. It was promoted through new GCSE and A-level computing qualifications (qualifications needed to progress to further education), supported by resources from the Raspberry Pi Foundation, Computing at School (CAS), and the National Centre for Computing Education (NCCE). Tools like Thonny and Mu Editor made it easier for teachers to introduce text-based programming without the steep learning curve of older languages.

Python's flexibility gave students a language that was both beginner-friendly and relevant to real-world software development. By the late 2010s, it had become the standard language for teaching programming in UK schools, effectively replacing both VB and the remnants of earlier BASIC dialects.

International comparisons

While the UK's shift to Python was driven by centralised curriculum reform, other countries have followed different paths to adopting Python, shaped by local policies, resources, and educational priorities.

United States

In the US, education is highly decentralised, so no national directive dictated programming language choices. However, Python has steadily gained popularity — especially in middle- and high-school electives and in introductory university courses, where institutions like MIT helped pave the way by switching

from Scheme to Python. Python is also supported in AP Computer Science Principles, although Java remains the official language for AP Computer Science A, meaning that both languages are commonly taught in parallel. In addition, JavaScript maintains a strong foothold, particularly in schools and programmes that emphasise web development pathways, reflecting the growing relevance of front-end programming and industry-aligned skills.

India

India's large and varied education system shows a strong trend toward Python in schools affiliated with national boards like CBSE (Central Board of Secondary Education), where Python has largely replaced C++ and Java in the senior secondary computer science curriculum. The language's accessibility and its alignment with real-world programming skills make it attractive for students aiming for careers in technology. The rise of online coding platforms and India's thriving edtech sector have both reinforced Python's dominance at both school and extracurricular levels.

Kenya

Kenya is in the midst of curriculum reform through its Competency-Based Curriculum (CBC), and while programming is still emerging in mainstream classrooms, Python is increasingly being taught in private and international schools, and through NGO-led initiatives. Programmes such as Africa Code Week, AkiraChix, and local Raspberry Pi Foundation outreach efforts have brought Python into after-school clubs and informal learning settings.

South Africa

In South Africa, high-school learners traditionally studied Delphi in information technology courses. However, there's a growing push to modernise the curriculum, with Python being introduced in pilot programs and enthusiastically adopted in extracurricular coding clubs. Several South African universities, including the University of Cape Town, have moved to Python for first-year programming courses — an influence that is filtering down into secondary schools. Local groups like Code4CT and GirlCode ZA are helping to expand access and support.

So is Python a good language to use?

Python is widely used in schools because of its clean, readable syntax and its ability to lower the barriers to entry for programming. For students encountering text-based code for the first time, Python allows them to focus on the logic of their programs without getting bogged down by complex syntax rules. It's often possible for a student to write and run their first program within minutes — a simplicity that builds early confidence and encourages experimentation. This is particularly valuable in secondary classrooms, where lesson time is limited and engagement is critical.

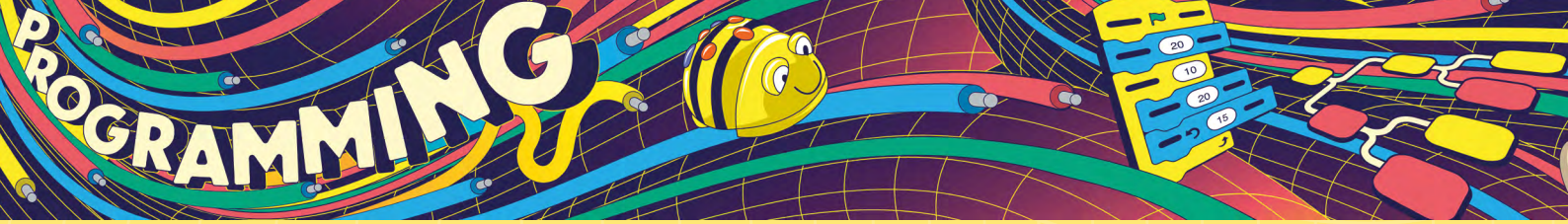
However, while Python's simplicity is its greatest asset, there comes a point at which Python's design limits learning. For instance, Python lacks true constants, making it difficult to demonstrate immutability. Python's dynamic typing hinders understanding of how different types of data are stored. Its lists differ from fixed-size arrays, preventing exposure to concepts like array bounds and memory layout. As an interpreted language, it bypasses the compile–run–debug cycle. Finally, Python's approach can confuse novices with self and flexible class models, complicating the explanation of encapsulation and inheritance.

In summary, Python is an outstanding language for teaching students how to think algorithmically and how to write clean, functional code. It is ideally suited for introductory programming and general problem-solving. However, when the curriculum shifts toward the mechanics of computation — such as constants, memory, type systems, and low-level data structures — Python's high level of abstraction can become a limitation. For this reason, many educators who teach older or more advanced students choose to supplement Python with exposure to languages like Java or C#, helping learners to gain a fuller understanding of both the theoretical and practical foundations of computer science. (HW)

Python has been used to build apps such as Instagram and Spotify

DIANE DOWLING

Diane is director of curriculum and resources at the Raspberry Pi Foundation, where she leads a team of talented educators creating resources for schools and Code Clubs.



UNLOCKING THE FUTURE

Java's latest features and why every student should know them

Have you ever tracked a package, streamed a movie, or ordered a ride-share? If so, you're using Java, one of the world's most trusted and widely used programming languages. Across a wide range of use cases, Java is powering the apps (helloworld.cc/java-apps) and services (helloworld.cc/java-uber) that shape our lives. Bottom line, the world runs on Java! Find more real-world examples at Learn.java.

Java's versatility, robustness, security features, and ongoing evolution make it an ideal choice for students preparing for the tech workforce. There is tremendous opportunity as Java becomes increasingly integrated with AI services and tools. With new features released every six months, Java is more powerful — and more accessible — than ever.

Java for today's learner

If you haven't seen Java since the early 2000s, you'll be amazed by its modern features. From simplified syntax for beginners to robust new tools for seasoned professionals, Java is designed to help you learn coding, solve problems, and build your future. AI code assistants benefit greatly from Java thanks to the large amount of open-source Java code and its long history of evolution to train upon. Even the most basic and free large language models will happily help you code in Java.

Java is the language of choice for many introduction to programming courses and the majority of data structures courses. For example, the AP Computer Science A course and exam, administered to nearly 95,000 students annually, uses Java as its language of choice.

Learn.java for teachers and students

To support students and teachers, we're excited about our recent launch of **Learn.java** — a vibrant new platform with hands-on tutorials, lesson plans, inspiring developer stories, and a powerful Java Playground to experiment with code right in your browser. Whether you're an educator seeking fresh resources or a student eager to explore the world of programming, we hope **Learn.java** will be your place to get current information and stay up to date with all the advancements in Java and how they relate to education.

What's new with Java?

Java 25 was released in September 2025. Some features you might find engaging and impactful for your students are:

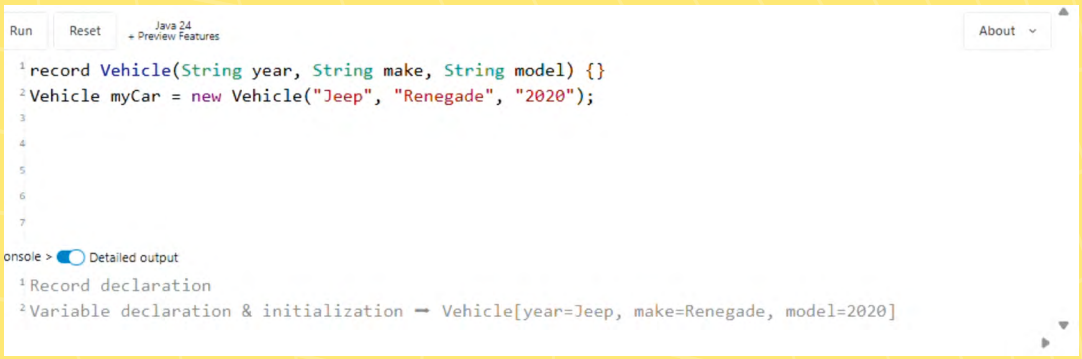
- Compact source files and instance main methods (Java 25)
- Records (Java 16)
- Switch expressions (Java 14)
- Pattern matching (Java 16, 21, 22, with preview features in 25)

For a complete list, review our Educator Briefings (**helloworld.cc/ed-briefing**).

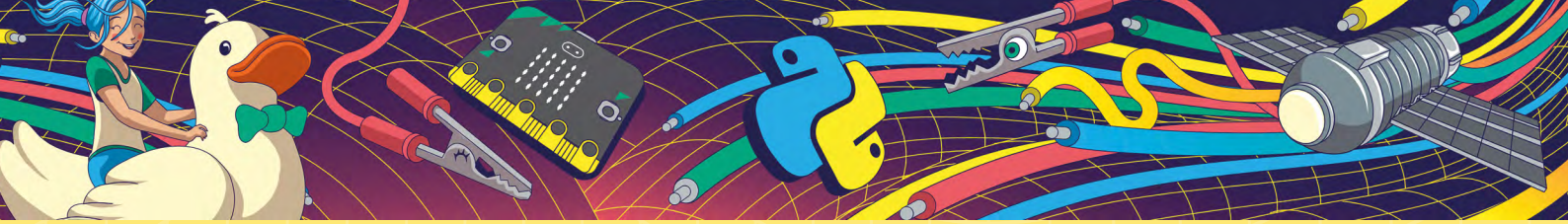
Compact source files and instance main methods

Instance main methods means your first program is just a few lines. Great for beginners and those who want to rapidly prototype or try code snippets, compact source files and instance main methods pave the on-ramp for new and experienced programmers to write smaller programs succinctly, without the need for constructs intended for programming in the large.

A new `IO` class simplifies input and output. This class is part of the `java.lang` package and is automatically made



An example of a new record type (Vehicle) and an object of this type being created using the Java Playground



BEFORE AND AFTER JAVA 25

HelloWorld.java > ...

```
1   public class HelloWorld {
        Run main | Debug main
2       public static void main(String [] args) {
3           System.out.println ("Java Rocks!");
4       }
5   }
6
7
8
```

Prior to Java 25

HelloWorld.java

```
    Run main | Debug main
1   void main(String [] args) {
2       IO.println ("Java Rocks!");
3   }
4
5
6
7
8
```

Java 25 and beyond

> JAVA'S VERSATILITY, ROBUSTNESS, SECURITY FEATURES, AND ONGOING EVOLUTION MAKE IT AN IDEAL CHOICE FOR STUDENTS

available. The static methods in this class are based on `System.out` and `System.in` and include `print(object obj)`, `println()`, `println(Object obj)`, `readln()`, and `readln(String prompt)`.

Writing your first program is much more accessible. Students can start writing programs without having to spend time learning a lot of new terms without context. Teachers no longer need to explain such things as what it means for a class to be `public` or a method to be `public` and `static`. The way `println` is called follows the same pattern students will use to call most methods in an introductory computer science course.

More top-level classes are being made available without needing to explicitly import them. Programs will automatically import the `java.base` module (**helloworld.cc/java-base**). This module includes `java.util` which makes working with data even easier by allowing the use of `ArrayList` without explicit import statements.

Records

Records let you store data simply and securely — no more writing boilerplate code. Released in Java 16, records allow us to treat data as data instead of objects. Records are simple to create and the right tool for dealing with data we don't want to change. The way they are designed protects the data from being altered, making it the safer and right solution.

Switch expressions

Switch expressions extends switch statements to be used as part of an expression. How are switch expressions different from switch statements?

- Switch expressions yield a value, typically assigned to a variable.
- Switch expressions use the keyword `yield` and do not require a `break` statement, as `yield` will return a value and the execution stops.
- Switch expressions require a semicolon (;) at the end.

Pattern matching

Code is more readable with pattern matching. Pattern matching checks the structure of an object and then extracts the data based on patterns. With pattern matching, the compiler recognises patterns in the code, making use of what it already knows is true and allowing more succinct statements to be written. Multiple pattern matching features have been released over several Java versions. There are currently four pattern matching features, with more to come in later releases.

Rediscover Java

Download the latest Java Development Kit (**java.com**) and discover Java's new tools like JShell, VS Code Extension, and all of the new features.

Find a new lesson plan, try a tutorial, or practise some code in the Java Playground at **Learn.java** today. (HW)

CRYSTAL SHELDON

Crystal is an experienced computer science teacher who is the former lead for AP Computer Science A. Through her role as Oracle's director of Java in Education, Crystal works to support students, teachers, and CS education providers using Java. She believes that providing students with authentic problems to solve makes computer science more engaging for all.

PROGRAMMING LANGUAGE CHOICE IN A-LEVEL PROJECTS

How to support students when they choose their programming project

Advanced level or A-level qualifications are subject-based qualifications that can lead to university, further study, or work. For most A-level computer science students (ages 17–18), the programming project is a major milestone. Worth around 20 percent of the final grade, it gives students the chance to take on a substantial, self-directed piece of work over several months. It's a rare opportunity to go beyond textbook exercises and instead tackle something of interest in the real world, manage a project, solve problems, design and test from scratch, and build their coding skills in the process.

One of the most important, and sometimes contentious, decisions students face early on is which programming language to use. The short answer is almost anything — but some clear favourites emerge.

What languages are students choosing?

Python: the undisputed leader in many schools, Python is familiar from earlier school stages, is easy to set up, and has a huge ecosystem of libraries and tutorials. Its simplicity makes it a safe choice for both teachers and students, and it can handle a wide range of project types, from games with Pygame to web apps with Flask.

JavaScript: JavaScript is popular with students building interactive websites, often pulling in HTML, CSS, and frameworks like React. Using JavaScript is a fast track to visually impressive results, but some teachers are wary of it, as it's not always aligned with exam specifications.

C#: C# is a strong structured language, and when paired with Unity, it becomes a gateway to 3D game development. But frameworks like Unity can abstract much of the 'hard stuff' that A-level projects are meant to demonstrate, reducing opportunities to show complexity.

PHP, C++, Swift, Rust, and others: PHP works well for database-driven web projects. C++ can be a good fit for hardware work. Swift and Rust tend to be niche choices, often driven by personal interest. With these, checking exam board rules is crucial, as not all languages are acceptable.

Whichever programming language our students choose, as teachers, we have to ensure that:

1. The language is appropriate for the student's skill level
2. We can support them in the language
3. It allows the project to meet required level of complexity without being so framework-heavy that the core logic is hidden

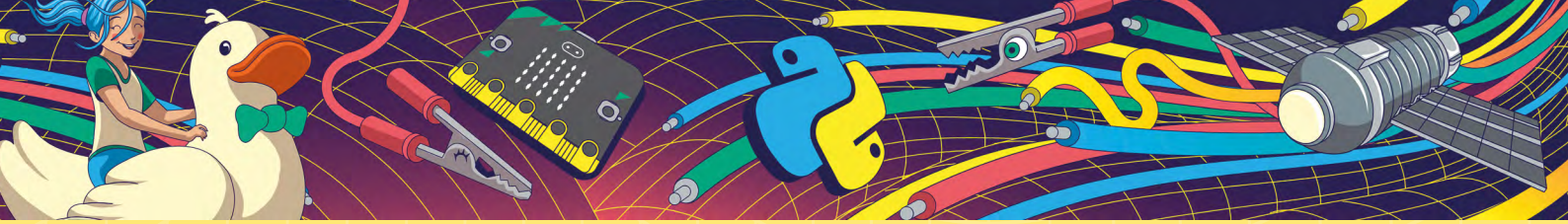
A-level computer science projects give students the freedom to choose their programming language, but that freedom comes with challenges for teachers

© Monkey Business/stock.adobe.com

Challenges for teachers

No one can be an expert in everything. To streamline support and marking, some teachers restrict projects to a single language; others allow freedom, but only to more capable students.

Adam Dimmick, head of digital learning and a computer science teacher at Reading School in England, explains his school's approach: "The vast majority of our students will use the main taught language for the bulk of their code — either Python or C# — but I've had some projects in Rust, Lua, Java, and Kotlin. Some use a combination, like Python for server-side and JavaScript for client-side. We typically expect students to demonstrate proficiency in the language



through some other (smaller) projects that they've completed previously. It's only ever our highest-ability students that seek to use a different language."

New syntax and unfamiliar error messages can make assessment slow and difficult. Worse, it's easy to miss plagiarism, especially when students follow step-by-step tutorials or copy AI-generated code. Students' enthusiasm can also lead to grand visions: 'I'll build a multiplayer 3D RPG!' Reality often hits when they run out of time or can't integrate the complex systems they planned. Finally, school IT systems can be the silent project killer. Installing new languages or frameworks can be difficult, especially if network administrators are cautious about perceived security risks.

Practical support strategies

Before projects begin, offer a shortlist of approved languages you know you can support and that exam boards accept. Run tech taster sessions early in the teaching year, showcasing different techniques in languages and frameworks that you are happy to support, such as connecting to a database, building a small object-oriented programming (OOP) game, or creating a basic web app. These are often good jumping-off points for students' project ideas. Help students set realistic project scopes, and explain how their language choice will impact complexity and assessment. Highlight skills, such as using OOP or database work, that they can transfer into exam skills.

Johanna Watkins, head of computer science at Highcliffe School in Christchurch, England, explains: "I let students choose their own programming language. The less confident usually stick with Python, while the more able often try something new before heading to university. Since the project work is independent, I'm not worried if I can't give hands-on help, especially when a language is a better fit for their idea. I point them towards reliable websites so they can teach themselves. In my experience, it's only the top students who step outside their comfort zone, and most have a pretty good sense of their own abilities."

During the project, encourage self-documentation. Students should be creating comments, development logs, and short write-ups explaining code choices and algorithms. These will be helpful for documenting the project and proving they wrote the code. Schedule checkpoints where students can demonstrate progress, or use GitHub commits which you can review.

Isabel Culmer, a computer science teacher at Barton Peveril Sixth Form College in Eastleigh, England, outlines her approach: "We do a 'viva'. We have a deadline for the technical solution. After that, we sit for 20 minutes with each student for them to talk us through their solution, and we give them a mark. We do this in lesson time when the rest of the class are writing up design, testing, and evaluation."

Conclusion

A-level programming projects are a brilliant opportunity for students to explore and innovate. But freedom without guidance can lead to frustration, incomplete work, and teacher burnout. The sweet spot is structured choice: give students the freedom to pick from a range of viable languages, help them set goals, and support them with general coding principles. Make them responsible for explaining and justifying their work, and you'll help them develop skills that matter in the real world. The lessons they learn will last much longer. (HW)

TEACHER TIPS: SUPPORTING A-LEVEL PROJECTS IN MULTIPLE LANGUAGES

A-level computer science projects give students the freedom to choose their programming language, but that freedom comes with challenges for teachers. Here's how to keep projects achievable and assessment manageable.

1. Curate the choices
Offer a shortlist of approved, exam-compliant languages you can support. Include pros and cons for each so students can make informed decisions.

2. Set realistic scopes
Guide students to match ambitions with their skills and the time available. Framework-heavy tools (such as Unity) can hide complexity, so ensure projects still meet marking criteria.

3. Build early foundations
Run taster sessions showing different languages and frameworks in small, achievable projects. Highlight transferable skills like OOP and database integration.

4. Encourage documentation
Ask students to log progress, comment on code, and explain design decisions. This helps with assessment and reinforces understanding.

5. Support without knowing it all
Focus on language-agnostic principles, logic, structure, and problem-solving, so you can guide even in unfamiliar syntax.

6. Use your network
Share resources with colleagues, and encourage peer mentoring among students.

With clear boundaries and collaborative support, students can explore creatively while staying within a framework that sets them up for success.

LAURA JAMES
Laura is a learning manager in the Ada Computer Science team at the Raspberry Pi Foundation. She is an enthusiastic computing educator with more than a decade of experience in secondary education.

PRIMM AND PROPER VIBE CODING

What happens when students bring ChatGPT, GitHub, and Canva into the classroom? **James Abela** looks at vibe coding with PRIMM and PROPER

JAMES ABELA
James is director of digital learning and entrepreneurship at Garden International School in Kuala Lumpur, Malaysia. He is founder of the South East Asia Computer Science Teachers' Association and **ReadySetCompute.com**.

I started vibe coding in September 2024 to make time-saving tools for teaching, including a Python display tool, a mindmap editor, and story dice (helloworld.cc/jsfun).

Vibe coding is an AI-enhanced approach to programming that prioritises natural language input, visual interaction, and fast iteration. Although the term 'vibe' is often used as shorthand, it was popularised by AI expert Andrej Karpathy in 2025 to describe the process of creating code through conversational, intuitive prompts, rather than traditional syntax. He describes it as: "I just see stuff, say stuff, run stuff, and copy-paste stuff, and it mostly works."

Tools such as ChatGPT, Google Gemini, GitHub Copilot, and Canva enable students to quickly generate code or designs based on plain-language descriptions. These platforms typically require users to be aged 13 or older, and teachers should check age restrictions and usage policies carefully before introducing them in the classroom.

When integrated with structured pedagogical approaches, vibe coding creates an engaging, inquiry-driven experience that supports both creativity and conceptual understanding.

The International Baccalaureate Middle Years Programme (MYP) encourages students aged 11–16 to explore learning through inquiry, creativity, and real-world connections. One of its core components is the MYP design cycle, a structured model used across subjects to guide students in solving problems through four key stages: inquiring and analysing, developing ideas, creating the solution, and evaluating (**helloworld.cc/myp-design-cycle**).

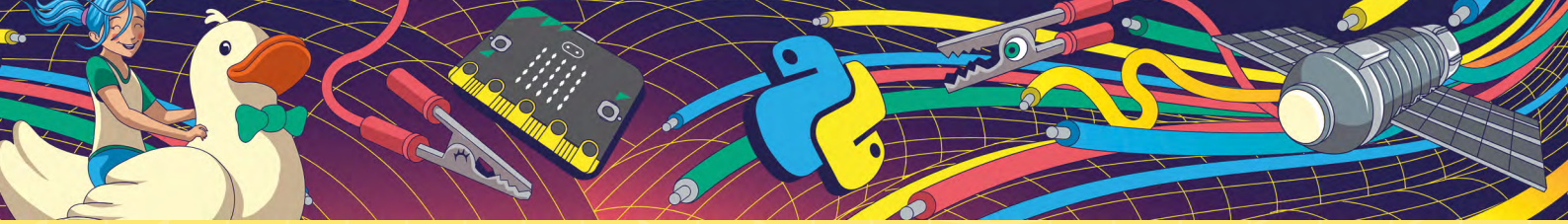
Teachers from across Malaysia came together to learn about vibe coding with Google Gemini and the Experience AI course (**experience-ai.org**)

Classroom structures

To guide and deepen the MYP design cycle experience, teaching frameworks such as PRIMM (Predict–Run–Investigate–Modify–Make) support learners in understanding and experimenting with code in a structured way (**helloworld.cc/qr-primm**). To complement PRIMM, I developed my own informal framework — PROPER — as a simple way to help students think more independently, especially when prompting AI tools.

PROPER stands for Purpose–Readability–Organisation–Patterns–Errors–Refinement. These prompts support students in asking thoughtful questions about the code they receive or generate. For instance, they might reflect on the Purpose of the code (what is it trying to achieve?), assess its Readability (can someone else understand it?), and consider how well it is Organised.

They then look for common Patterns, check for Errors, and think about how



AI PERFORMANCE

Source: OpenAI usage data and documentation

Language	AI support quality	Notes
JavaScript	Very good	Ideal for interactive web projects and beginner tasks. Frequently used in vibe coding.
Python	Very good	Highly readable and versatile. Works well for quizzes, chatbots, and data tasks.
Java	Good	Well-supported, but more verbose. Useful for structured, object-oriented examples.
C#	Good	Especially strong in Unity contexts. Works well with specific prompting.
C/C++	Moderate	Needs precise prompts. Can be unsafe without strong guidance.
Swift	Moderate	Good for simple iOS tasks. Less reliable for advanced use without clarification.
Low-level / niche languages	Inconsistent	Less exposure during training. May need manual correction.

the code could be Refined. This reflective process is especially valuable when using AI tools, as students must still evaluate and improve the suggestions provided.

Together, PRIMM and PROPER provide a structured yet flexible pathway for developing programming skills, such as within the MYP design cycle. Students might begin with JavaScript, where they can create interactive web-based projects and experiment with user input and animations.

As their skills grow, they can transition to Python to deepen their understanding of programming logic using loops, conditionals, and functions in projects such as score calculators or chatbots. In both languages, students are encouraged to reflect on their process, make informed changes, and articulate how and why their code works.

Prototyping with AI

One of the most powerful applications of vibe coding is for rapid prototyping — quickly developing testable versions of programs or designs. This aligns closely with the 'developing ideas' and 'creating the solution' stages of the MYP design cycle. For example, prompting ChatGPT with 'Write a Python program for a number-guessing game' can generate a draft ready for analysis and refinement.

Canva can similarly support prototyping in web design or visual communication tasks. When guided by PRIMM and PROPER, this process helps students test, explore, and improve their work through structured inquiry and iteration — mirroring practices that are used in real-world software and product development.

Vibe coding encourages a shift from passive learning to active experimentation. It invites students to engage critically with tools, develop a better understanding of programming concepts, and experience immediate feedback. When used together with intentional scaffolding such as PRIMM and PROPER, it becomes more than just a way to generate code — it becomes a springboard for inquiry, creativity, and thoughtful reflection.

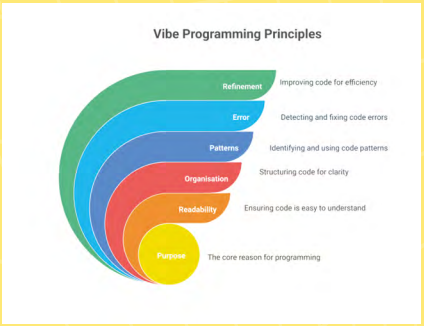
AI performance

While vibe coding makes it easy to get started across many languages, some languages are handled more effectively by AI tools than others. The table on the left provides a general overview of how ChatGPT currently performs with different programming languages in educational contexts.

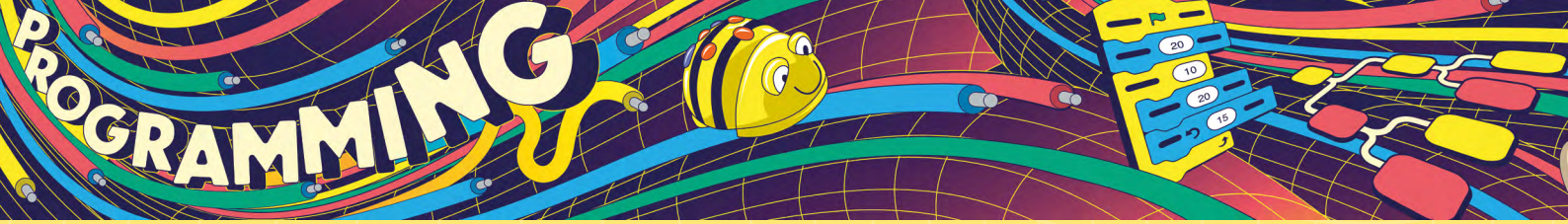
Vibe coding, while engaging and accessible, has clear limitations that educators must consider. Within just three to five prompts, AI-generated code can begin to degrade — introducing bugs, removing original features without explanation, or hallucinating solutions that do not work. These issues become more frequent in larger projects. In my experience, reliability often declines after about 300 lines of code. Without a firm foundation in programming concepts, students may struggle to troubleshoot or understand what their code is actually doing.

Yet despite these challenges, vibe coding remains a valuable entry point, particularly when paired with frameworks such as PRIMM and PROPER. These structures help learners move beyond surface-level interactions to develop more resilient, purposeful code.

When used well, vibe coding can spark curiosity, support creative problem-solving, and lay the groundwork for deeper learning. It offers a clear opportunity to introduce core programming concepts in a way that feels relevant, empowering, and adaptable to a range of learners. (HW)



The PROPER framework helps encourage students to reflect on the code they produce when vibe coding



OFFLINE PROGRAMMING IN THE AGE OF AI

How every student can learn computational thinking and AI concepts, regardless of access to the internet

The digital divide became starkly apparent during the Covid-19 pandemic, when many of my students lost access to online learning resources. I understood the challenge first-hand. As a first-generation immigrant, I learnt programming as an English Language Learner in rural central Pennsylvania, USA, where internet service was available but often unaffordable for families. While districts scrambled to provide connectivity and resources during the pandemic, I focused on a different approach: delivering offline education, using technology which didn't require internet access.

I work at a regional education service agency, the Tuscarora Intermediate Unit (**tiu11.org**), which supports nine rural school districts in Pennsylvania. During this time, we deployed Kolibri on Raspberry Pi devices. Kolibri (**learningequality.org**), developed specifically for disconnected learning environments, became our solution for reaching students whose families couldn't afford reliable internet service. This enabled students without internet access to receive the same resources and materials as students with internet access. These small computers became educational hubs, delivering content and assessment directly to student devices through local networks.

One of our projects involved developing a Python course that taught students to program flight patterns for Parrot Mambo drones — all delivered through Kolibri's offline platform. Students learnt programming fundamentals while controlling real hardware and building software with real-world applications; programming drones was especially pertinent during the pandemic when they were used to deliver medicine in remote areas.

The results challenged my assumptions about online versus offline learning. Students engaged more deeply with programming concepts when they weren't competing with internet distractions or worried about data usage. They spent more time debugging code and understanding logic rather than copying solutions from online forums. Most importantly, students in rural areas with limited financial resources could participate equally in computational thinking activities.

This experience also highlighted a critical gap in computer science education. When students get stuck on programming problems, whether debugging code, understanding syntax errors, or learning new concepts, their first instinct is to search online. They expect to google error messages or watch YouTube tutorials. However, the students I serve often live in areas with limited or no internet access. These students face a fundamental learning barrier.

Computational thinking without connectivity

Offline programming naturally emphasises logic over tools. Without internet access, students can't rely on copying code snippets or following tutorials mindlessly. They must understand loops, conditionals, and functions at a fundamental level. Our local Python installations paired with Parrot Mambo drones provided hands-on learning, where students programmed real flight patterns, immediately seeing the results of their logical thinking.

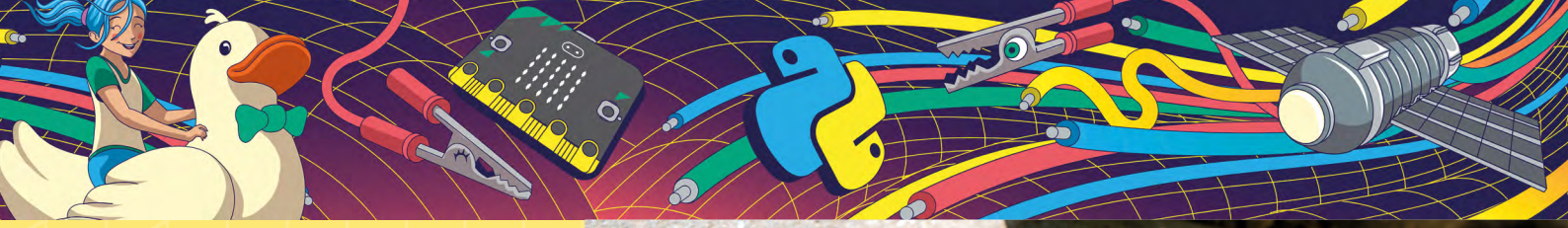
The Raspberry Pi's GPIO pins opened additional possibilities for physical computing. Students connected sensors and LEDs directly to create weather monitoring systems, which we later utilised for monitoring hydroponics and aquaponics. These projects taught programming concepts through tangible outcomes — a blinking LED confirmed their conditional statements worked and sensor readings validated their data-processing algorithms.

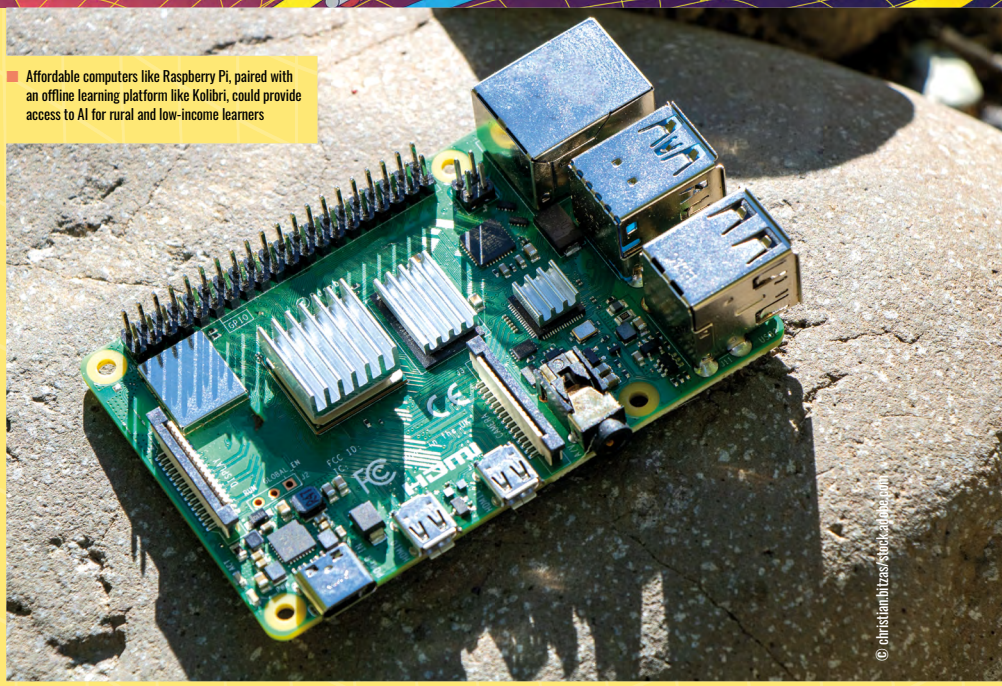
Using this offline method, students demonstrated understanding through working code that controlled actual hardware rather than selecting answers from a formative assessment. Without online forums to copy from, project-based learning flourished.

Most importantly, this approach prepares students to collaborate with AI rather than creating dependency. When we eventually introduced AI tools in school, students who understood programming fundamentals through offline practice grasped machine-learning concepts more readily. They approached AI platforms as tools to enhance their problem-solving rather than as magic solutions.

Local AI tutors

While offline programming addresses the digital divide, a new challenge emerges: the AI access divide. Currently, AI-powered educational tools require internet connectivity and subscription fees, creating





Affordable computers like Raspberry Pi, paired with an offline learning platform like Kolibri, could provide access to AI for rural and low-income learners

© christianbitzas/Stockagence.com

additional barriers for disconnected learners.

The numbers highlight the problem. Only 57 percent of families earning less than $30,000 annually have internet access at home (**helloworld.cc/us-internet**). Rural internet penetration remains at 73 percent compared to 86 percent in suburban areas. For families who do access the internet, AI tools present additional financial barriers. ChatGPT Plus costs $20 a month, as do Claude Pro, Perplexity AI Pro, etc. Free users cannot opt out of data training, requiring paid plans for privacy protection.

This creates a two-tiered system where AI-enhanced education becomes another privilege for those who can afford connectivity and subscriptions.

> OFFLINE LEARNING PREPARES STUDENTS TO WORK WITH, RATHER THAN DEPEND ON, AI

My current exploration focuses on deploying lightweight language models directly on Raspberry Pi devices alongside Kolibri. Models like Microsoft's Phi-3 and Phi-4, and Google's Gemma show promise for running locally on Raspberry Pi 4 and Raspberry Pi 5 hardware, while teaching computer science concepts aligned with CSTA standards (**helloworld.cc/csta-standards**).

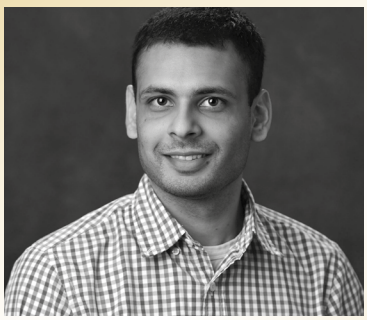
The vision is to provide a $75 Raspberry Pi that offers 24/7 AI programming tutoring, eliminating internet dependency and subscription costs. Students could receive personalised coding assistance, debugging help, and concept explanations while working through offline exercises. The same device serving Kolibri content could simultaneously run an AI tutor, creating a complete offline-learning ecosystem.

The potential impact extends beyond individual learning. Rural schools could deploy these systems to provide AI-enhanced computer science education without requiring reliable internet or ongoing costs. Libraries and community centres could offer AI-powered programming assistance to entire communities. Most importantly, students in economically disadvantaged areas could access the same AI-enhanced learning experiences available to their connected peers.

This addresses a fundamental equity issue. Rural students and low-income families are unable to access AI-enhanced education due to barriers created by limited internet connectivity and high subscription costs. The result is that future developers will come primarily from connected, affluent communities. Offline AI tutoring changes this dynamic. A student in rural Pennsylvania gains access to the same programming assistance available to peers in well-connected areas. When we include diverse voices in CS education — students from farming communities, immigrant families, and economically disadvantaged backgrounds — we build stronger technology solutions for everyone.

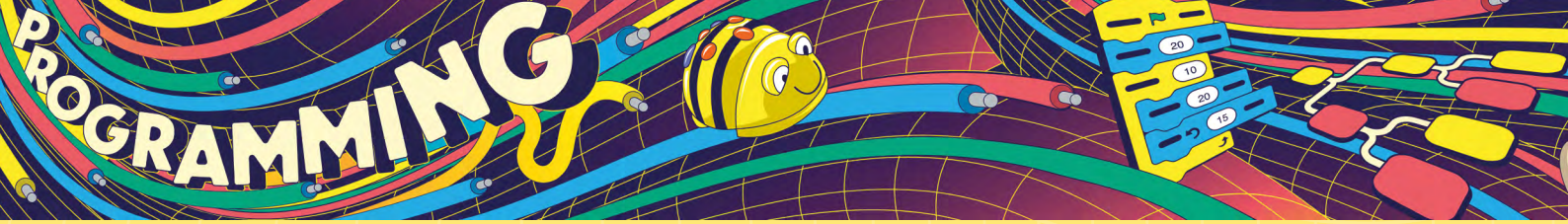
Conclusion

The future of computer science education must include pathways for economically disadvantaged learners and those in rural areas. By building offline programming curricula and exploring localised AI deployment, we ensure every student develops computational thinking skills necessary for an AI-integrated world. Sometimes, the most inclusive approach to cutting-edge education involves stepping back from connectivity requirements and focusing on fundamental skills that transcend technological barriers. (HW)



JIGAR PATEL

Jigar is the director of innovation & special projects at Tuscarora Intermediate Unit 11 and vice president of the CSTA Susquehanna Valley Chapter. He is also a Raspberry Pi Certified Educator and an alumnus of the CSTA Equity Fellowship.



TEACHING AND LEARNING PROGRAMMING

Educators reflect on the dual challenge of learning to program while teaching programming to students

For traditional subjects like maths or science, most K–12 teachers have years of exposure, both as students and through formal training within their teacher preparation programme. Computer science is a relatively new addition to the curriculum, and here in Iowa in the United States, many K–12 teachers are stepping into the role of computer science instructors with little to no prior experience in the subject. This creates a unique and often daunting situation: teachers are learning the content themselves while simultaneously teaching or preparing to teach it.

Working with teachers placed in this situation has given us a unique opportunity to consider the intersection between the teaching and learning of programming by asking these teachers to reflect on the often-overlooked elements of programming. This article summarises the most common takeaways mentioned by these teachers as they reflect on their learning, and how they will translate to their own classrooms.

Background

In order to better prepare in-service teachers to become computer science teachers, our programme offers an 18-month cohort-based programme where groups of teachers from across Iowa complete five courses. Upon completion of the programme, they can apply for the state's computer science endorsement on their teaching licence (**helloworld.cc/diesburg**). Since the summer of 2023, our programme has worked with over 220 teachers. As demonstrated by our cohort from 2024, they come from a surprisingly wide variety of grade bands (**Figure 1**), disciplines (**Figure 2**), experience levels (**Figure 3**), and comfort levels (**Figure 4**).

Within this programme, there are two courses that help participants broaden their skills as teachers of programming. The first course, 'Fundamentals of Programming', begins with five weeks of Scratch, where they learn the foundational vocabulary and structure of programming. They then transition to ten weeks of problem-solving using Python. In the second course, 'Teaching and Learning of Programming' (TLP), they continue their journey as programmers while considering teaching techniques in the context of both their learning journey and their own classrooms. This course uses *The Big Book of Computing Pedagogy* (**helloworld.cc/bbcp**) as its main textbook — you can read more about our choice to use this resource at **helloworld.cc/schafer**.

As part of the wrap-up for the TLP course, participants respond to writing prompts asking them to identify the key concepts and skills of programming as well as what they

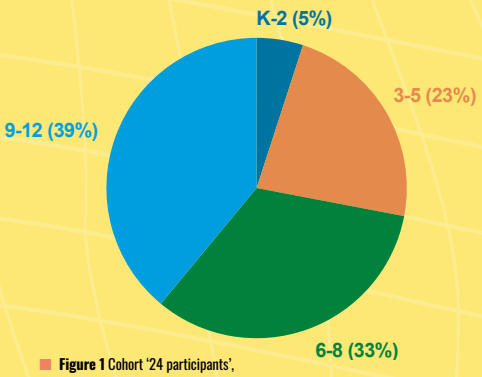


Figure 1 Cohort '24 participants', grades taught

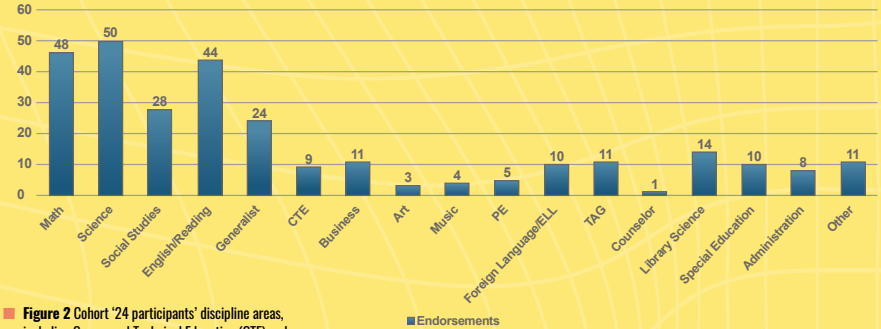


Figure 2 Cohort '24 participants' discipline areas, including Career and Technical Education (CTE) and Talented and Gifted (TAG) endorsements

Preparing in-service teachers to become computer science teachers

have learnt about teaching programming. The final prompt asks them to reflect on their journey as a programmer and to identify the biggest takeaways from their learning over the previous two courses. While this question was meant as a simple wrap-up of the previous writing prompts, participant responses have been incredibly insightful. Here are some of the major points, from both a learner perspective and a pedagogical/teaching perspective.

Learner perspective

While some teachers had prior coding experience, many had none. The following are some of their takeaways:

- **Even if your end goal is to learn a text-based programming language, it can help to start with something visual like Scratch.** Many of our teachers appreciated starting with Scratch in their programming journey. One teacher said, "When I started on this adventure last summer, I was someone who had never tried anything with programming or coding. I was a novice programmer at best, but even that is a stretch. I loved the block coding that we started with. It was a great way for me to feel like I had actually accomplished something." Others found it reduced initial intimidation and stress in learning programming: "I appreciated that last semester started off with learning Scratch. I feel like that helped us learn the basics of programming without the stress and worries about the syntax, like remembering to include a semicolon here or a comma there." Finally, other teachers found it was good for seeing abstract concepts in that it "helped me to visualise the program and perform code tracing before I realised what code tracing was".
- **While the initial learning journey can be rife with frustration when something doesn't work, confidence can be built in a variety of ways.** For some, it's a shift in mindset around the unknown. One wrote, "While technical skills are essential, I've come to understand that learning to program is just as much about developing problem-solving confidence and learning to be comfortable with not knowing something — yet." One teacher recognised that they needed to combat feelings of not being good enough, and so it's important to cultivate an environment in which "mistakes are seen as part of the learning process rather than signs of failure". Another teacher stated, "Struggling with bugs or hard problems isn't a sign of failure, it's a normal and expected part of programming." One teacher summed it up nicely: "Productive struggle is where the greatest learning occurs."
- **Don't overlook the small wins.** The ability to apply learnt concepts to solve real-world problems, even small ones, can significantly contribute to growth. One teacher shared, "When I get even a small block of code to work, it helps me to see that I'm moving in the right direction." These small wins can be even more impactful if they are used to solve small, practical problems of personal interest. Two examples from teachers included using Python to sort data for a fantasy baseball draft, and using it to organise and analyse what they do for a bakery business.
- **It's important to understand the problem before you code.** There is a tendency, especially with graphical and visual languages such as Scratch, to start to code through trial and error. Many of our teachers admitted that it took them a while to realise that planning the solution before starting was an essential part of the process. One teacher summed it up nicely: "Planning has also become a bigger part of my process. Before starting to write any code, I've learnt

> TEACHERS ARE LEARNING TO PROGRAM THEMSELVES WHILE TRYING TO TEACH IT

> TEACHERS WHO ARE LEARNING TO PROGRAM CAN RELATE TO THEIR STUDENTS

BEN SCHAFER

Ben is a professor of computer science at the University of Northern Iowa, as well as coordinator of their computer science education programme. If he could figure out how to do that from a kayak, he would.

SARAH DIESBURG

Sarah is an associate professor of computer science at the University of Northern Iowa and a proud member of the CS education team. She is currently trying to master the algorithms for playing the guitar.

to take the time to sketch out what I want to achieve, what the user needs, and how each step should logically follow the next. This isn't just about avoiding mistakes — it's about setting a solid foundation so that I don't have to backtrack later. The more thought I put into the planning process, the smoother the coding and debugging stages are. As we've learnt, jumping straight into coding without a plan can lead to confusion and unnecessary errors."

- **Debugging and testing code are important, too.** The shift from seeing debugging as a chore to an essential learning experience is important. As one teacher put it, "Almost all the code I wrote during the previous class did not work the first time; I had to debug and fix every program I wrote. This debugging and correcting my mistakes is where I learnt the most about coding." However, debugging goes beyond just getting the code to work. Testing the code is also where the learning takes place. As one teacher shared, "I have grown as a programmer, because I have eliminated the misconception that just because the program runs, it is correct. This was a hard concept for me to come to grips with; additionally, it's harder for students to understand."

Pedagogical and teaching perspective

Teachers considered teaching techniques both in the context of their learning journey and what would work in their classrooms:

- **Don't overlook teaching strategies that involve reading code before writing code.** Teachers thought strategies like PRIMM (Predict–Run–Investigate–Modify–Make; **helloworld.cc/primm**) and Parson's Problems (**helloworld.cc/parsons-problems**) were helpful for building program reading comprehension and for understanding problem-solving while providing scaffolding. One teacher shared, "I've also started using PRIMM in my own classroom. It gives my students a clear structure to follow, which helps build their confidence. Instead of feeling like they must understand everything at once, they can focus on just one part at a time."
- **Not all activities need to be done at the computer.** Many teachers saw the value of unplugged activities, where students can get up, move around, and

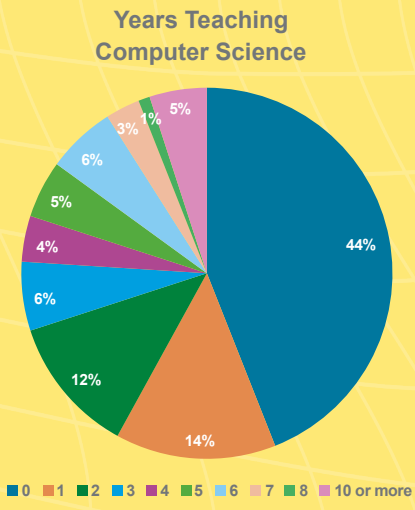


Figure 3 Cohort '24 participants', years teaching computer science

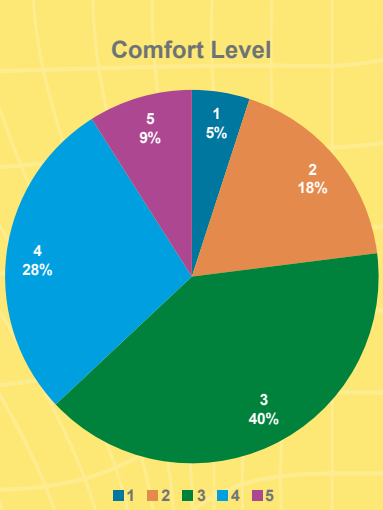


Figure 4 Cohort '24 participants', comfort level with computer science from low (1) to high (5)

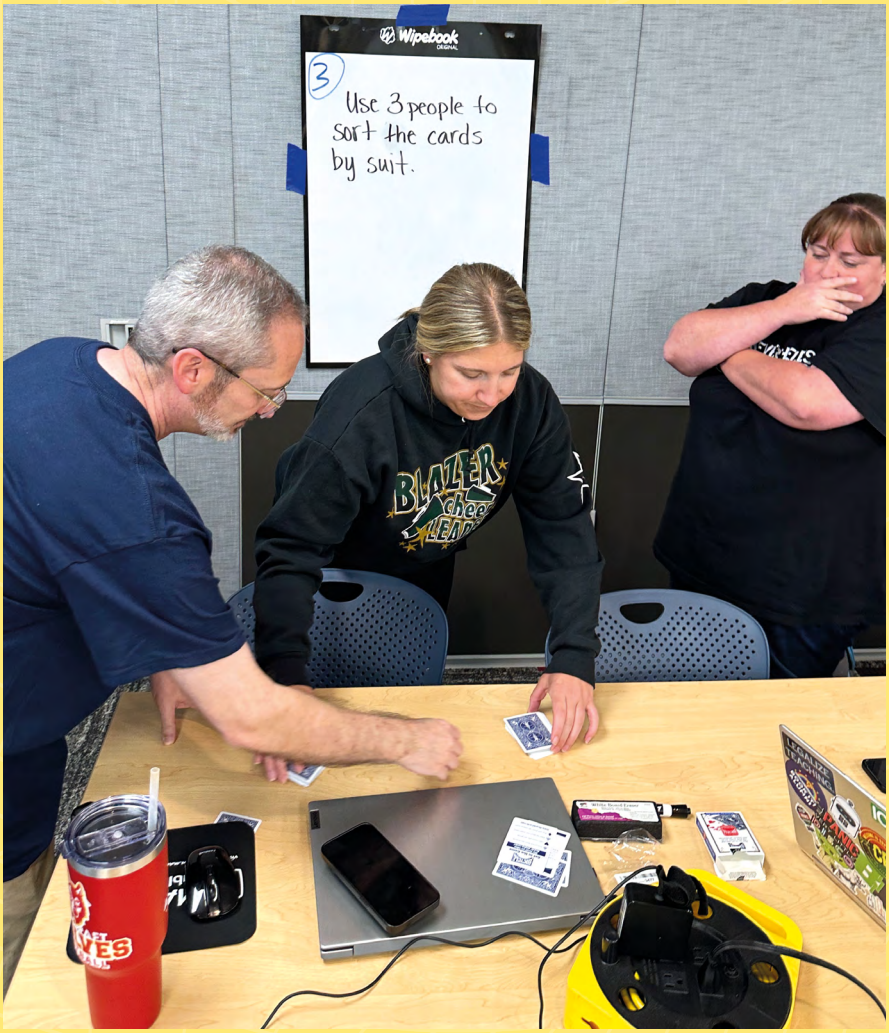
FURTHER READING

Diesburg, S., Schafer, J. B., & Morrison, B. B. (2025, February). Curriculum for a Comprehensive Statewide In-Service CS Teacher Training Program. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1* (pp. 255–261) (**helloworld.cc/diesburg**)

Schafer, B. & Rogers, C. (2024). USA: Training Computer Science Teachers. *Hello World 23,* 23–24 (**helloworld.cc/schafer**)

learn programming topics by solving problems away from the computer. One teacher had this to say about the use of unplugged activities for first-time coders: "My students understood the concept much better than when they only saw it on a screen or listened to me talk about it. I realised that not all students learn best by typing code; some need to move, talk, and interact. Unplugged activities help reach those learners."

- **Observing live coding demonstrations helped teachers to both learn to program and to plan to use this technique in their classrooms.** One teacher shared, "I did more live coding and live debugging with my students around the concept of variables, as the [*Big Book of Computing Pedagogy*] suggested. I saw a significantly higher number of students successfully using variables, with much less confusion about their purpose and importance, compared to my past students." However, it's important to note that making mistakes is also OK! Another teacher noted, "Seeing the emphasis on live coding and worked examples has helped me embrace a more iterative, mistake-friendly approach to learning and teaching code."
- **Learning should not always be done alone, and collaboration with peers can be essential to the learning process.** Pair programming and small group discussions, such as with Process-Oriented Guided Inquiry Learning (**pogil.org**), can be very helpful. One participant observed, "Working together allows students to piece together information and solidify their definition of a concept. Students also learn to share the cognitive load. When students work in pairs, one student can focus on the problem and the other can work on the implementation."
- **Code quality matters.** A lot of the teachers remarked on the importance of good code (clean, efficient, readable) versus smelly code (inefficient, unclear code). One teacher put it this way: "I spend time optimising the way my workshop is set up, the way my office is set up, the way my classroom is set up. If I didn't do that, my shop, office, and classroom would still work. I could still get the same stuff done, but it would be extremely inefficient. I'd constantly run into inefficiencies that would drive me nuts and make my jobs harder. Seeing this concept officially taught [in the context of programming] was great."



Teachers trying out unplugged activities

Conclusion

Being new to programming when teaching programming to students can be a strength. As one teacher shared, "The biggest benefit of learning a new programming language is that I feel like I can better relate to what my students are going through as they are learning to code as well." As professors, we appreciated the candid reflections of our teachers and are happy to share these valuable takeaways with Hello World! (HW)

INTEGRATING GENERATIVE AI INTO INTRODUCTORY PROGRAMMING CLASSES

Bonnie Sheppard explores a course which encourages us to work with, and not against, GenAI tools

Generative AI (GenAI) tools like GitHub Copilot and ChatGPT are rapidly changing how programming is taught and learnt. These tools can solve assignments with remarkable accuracy. GPT-4, for example, scored an impressive 99.5 percent on introductory programming exams, compared to OpenAI Codex's 78 percent just two years earlier (helloworld.cc/denny). With such capabilities, researchers are shifting from asking, 'Should we teach with AI?' to 'How do we teach with AI?'

Leo Porter and Daniel Zingaro have spearheaded this transformation through their groundbreaking undergraduate programming course at UC San Diego, USA. Their innovative curriculum integrates GenAI tools to help students tackle complex programming tasks while developing critical thinking and problem-solving skills.

Leo and Daniel presented their work at a Raspberry Pi Foundation research seminar in December 2024. During the seminar, it became clear that their insights have particular relevance for teachers in secondary education thinking about using GenAI in their programming classes.

Practical applications

In 2023, Leo and Daniel introduced GitHub Copilot in their introductory programming course, with large language models at its centre. The course included creative, open-ended projects that allowed the 552 participating students to explore their interests while applying the skills they'd learnt. The projects covered the following areas:

- **Data science:** students used Kaggle data sets to explore questions related to their fields of study — for example, neuroscience majors analysed stroke data. The projects encouraged interdisciplinary thinking and practical applications of programming.
- **Image manipulation:** students worked with the Python Imaging Library to create collages and apply filters to images, showcasing their creativity and technical skills.
- **Game development:** a project focused on designing text-based games encouraged students to break down problems into manageable components while using AI tools to generate and debug code.

Students consistently reported that these projects were not only enjoyable but also responsible for deepening their understanding of programming concepts. A majority found the projects helpful or extremely helpful for their learning.

Core skills for programming

Leo and Daniel emphasised that teaching programming with GenAI involves fostering a mix of traditional and AI-specific skills (**Figure 1** provides an example of this). The approach for their course centred on six core competencies:

- **Prompting and function design:** students learnt to articulate precise prompts for AI tools, honing their ability to describe a function's purpose, inputs, and outputs, for instance. This clarity improved the output from the AI tool and reinforced students' understanding of task requirements.
- **Code reading and selection:** AI tools can produce any number of solutions, and each will be different, requiring students to read and evaluate the options critically. Students were taught

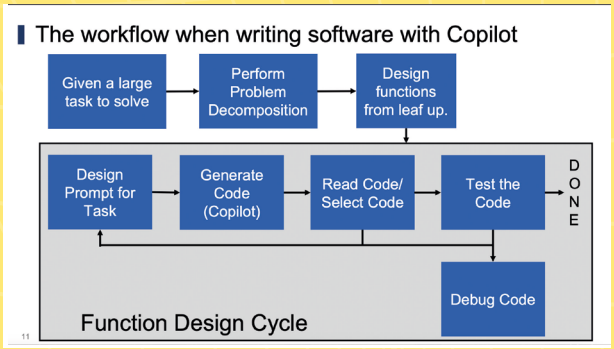


Figure 1 Writing software with GenAI applications needs to be approached differently to traditional programming tasks

FURTHER READING

Vadaparty, A., Zingaro, D., Smith IV, D. H., Padala, M., Alvarado, C., Gorson Benario, J., & Porter, L. (2024). CS1-LLM: *Integrating LLMs into CS1 Instruction. In Proceedings of the 2024 on Innovation and Technology in Computer Science Education v. 1* (pp. 297–303). (**helloworld.cc/vadaparty**)

Porter, L. & Zingaro, D. (2024) *Learn AI-Assisted Python Programming* (2nd ed.). Manning. (**helloworld.cc/porter**)

Porter, L. & Zingaro, D. (2023). *Courseware for CS1 courses that incorporate LLMs* [Data set]. GitHub. (**helloworld.cc/cs1-llm**)

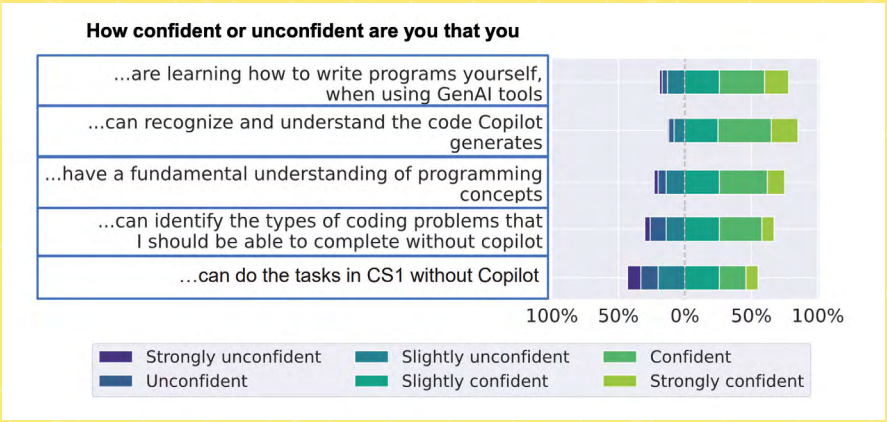
to identify which solution was most likely to solve their problem effectively.

- **Code testing and debugging:** students practised open- and closed-box testing, learning to identify edge cases and debug code using tools like doctest and the Visual Studio Code debugger.
- **Problem decomposition:** breaking down large projects into smaller functions is essential. For instance, when designing a text-based game, students might have separated tasks into input handling, game-state updates, and rendering functions.
- **Leveraging modules:** students explored programming domains and identified useful libraries through interactions with Copilot. This prepared them to solve problems efficiently and creatively.
- **Ethical and metacognitive skills:** students engaged in discussions about responsible AI use and reflected on the decisions they make when collaborating with AI tools.

Adapting assessments for the AI era

The rise of GenAI has prompted educators to rethink how they assess programming skills. In Leo and Daniel's course, traditional take-home assignments were de-emphasised in favour of assessments that focused on process and understanding. The researchers chose several types of assessments — some involved having to complete programming tasks with the help of GenAI tools, while others had to be completed without. For example, quizzes were used to evaluate students' ability to read, test, and debug code — skills critical for working effectively with AI tools. Final exams included both tasks that required independent coding and tasks that required Copilot. Students also submitted projects alongside a video explanation of their process, emphasising problem decomposition and testing. This approach highlighted the importance of critical thinking over rote memorisation.

Figure 2 Survey results



Challenges and lessons learnt

While Leo and Daniel reported that the integration of AI tools into their course has been largely successful, it has also introduced challenges. Surveys revealed that some students felt overly dependent on AI tools, expressing concerns about their ability to code independently (**Figure 2**). Addressing this will require striking a balance between leveraging AI tools and reinforcing foundational skills.

Additionally, ethical concerns around AI use, such as plagiarism and intellectual property, must be addressed. Leo and Daniel incorporated discussions about these issues into their curriculum to ensure students understand the broader implications of working with AI technologies.

A future-oriented approach

Leo and Daniel's work demonstrates that GenAI can transform programming education, making it more inclusive, engaging, and relevant. Their course attracted a diverse cohort of students, as well as students traditionally underrepresented in computer science — 50 percent of the students were female and 66 percent were not majoring in computer science — highlighting the potential of AI-powered learning to broaden participation in computer science.

By embracing this shift, educators can prepare students not just to write code but to also think critically, solve real-world problems, and effectively harness the innovations in AI which are shaping the future of technology. (HW)

BONNIE SHEPPARD

Bonnie is the programme coordinator in the research team at the Raspberry Pi Foundation. A former secondary-school teacher and lecturer, she is interested in translating research into practice.

CPD opportunities helped me to find my people

BIT BY BIT

How CPD upgraded my approach to programming instruction

When I first began teaching programming, I'll admit I was intimidated. Not by the students, not by the content, but by the pace of change in computer science education. The world of programming evolves faster than most subjects, and staying current felt like trying to catch a moving train. I knew that to be effective and confident in the classroom, I would need to commit to continuous professional development (CPD). What I didn't realise at the time was just how transformative that journey would be — not just for my teaching practice, but for my own identity as an educator.

Getting uncomfortable

My first serious dive into professional learning came through a summer coding bootcamp. I went to sharpen my Java and Python skills, but gained much more: a community of educators who were as hungry as I was to learn and grow. Surrounded by teachers from across the United States, I realised that we all shared similar fears, doubts, and aspirations. We leaned into the challenge, encouraged each other, and exchanged practical strategies we could take back to our classrooms.

That experience opened my eyes to what CPD could really offer — not only content knowledge but also confidence, connection, and purpose.

Pathfinders and the power of community

One of the most pivotal moments in my CPD journey was attending the Pathfinders Summer Institute (**helloworld.cc/pathfinders-summer-institute**). At the time, I still carried a bit of imposter syndrome with me, wondering if I truly belonged in rooms filled with 'real' computer science educators. But Pathfinders flipped that narrative.

At Pathfinders, I participated in hands-on workshops that weren't just about syntax or tools. They were about pedagogy, equity, and how to design learning experiences that ignite curiosity. I learnt how to scaffold programming challenges, how to use unplugged activities to build conceptual understanding, and how to integrate storytelling into coding lessons to make abstract concepts more relatable and concrete. More importantly, I met mentors and peers who became part of my ongoing support network.

I returned to my school not just with lesson plans but with a new mindset: that I could be both a learner and a leader in CS education.

Finding my people

The professional growth didn't stop there. Attending the CSTA (Computer Science Teachers Association; **csteachers.org**) Annual Conference was a turning point. The sessions were both informative and empowering. I connected with educators from all over the country, participated in deep dives into Python, AI, and game development, and walked away with a renewed sense of purpose. At CSTA, I found my people — teachers who understand the challenges and joys of teaching code.

Then came Indiana CSPDWeek 2025, a professional development experience unlike any other (**helloworld.cc/cspd-week-2025**). It was there that I participated in a session titled 'Introduction to Python programming', which not only reinforced my technical skills but also gave me fresh insight into how to bring coding to life for students aged 11 to 18. We created lessons, collaborated on projects, and even made commercials to pitch our ideas to other educators. It was energising, practical, and empowering, all at once.

Both of these experiences helped me realise that I didn't have to do this work alone. There's a whole community of educators out there ready to share ideas, provide support, and cheer each other on.

Fellowship as a catalyst

Being selected for a national CS education fellowship was a game changer (**helloworld.cc/csta-impact-fellow**). As a CSTA IMPACT Fellow, I had the opportunity

to engage in long-term professional development, coach newer teachers, and contribute to state and national conversations around CS education. The fellowship made me a better teacher as well as an advocate.

Through the fellowship, I co-facilitated workshops, helped review curriculum materials, and even shared my story with other educators who, like me, once felt unsure about their place in this field. These leadership opportunities pushed me out of my comfort zone and helped me see the bigger picture: teaching programming isn't just about loops and functions — it's about preparing students for a world shaped by technology.

Co-facilitating workshops and sharing is a great way to push you out of your comfort zone

From professional learning to classroom practice

All of these experiences — bootcamps, conferences, and fellowships — had a direct impact on my classroom. Here's how:

1. **Student-centred learning:** CPD taught me to move away from 'sage on the stage' teaching. Instead of lecturing, I now facilitate learning. My students engage in pair programming, collaborative projects, and debugging challenges that build both technical and soft skills.
2. **Iterative thinking:** professional development shifted my mindset from perfection to progress. I encourage students to see failure as feedback and to iterate on their code like real developers do. We celebrate productive struggle in my classroom now.
3. **Tech integration with purpose:** before CPD, I often introduced tools just because they were popular. Now, I'm more intentional. If a tool doesn't deepen understanding or foster creativity, I leave it out. Tools like Tinkercad, Scratch, or Code.org are integrated only when they serve clear learning outcomes.
4. **Confidence as a STEM leader:** most importantly, CPD helped me believe in myself. I'm no longer afraid to say 'I don't know', and I model curiosity alongside my students. I've become the go-to person in my building for STEM integration — not because I know everything, but because I'm always willing to learn.

Investing in my future

One of the greatest outcomes of these experiences is that they inspired me to take the next big step in my career: I am currently pursuing my master's in computer science education. As I continue to learn about pedagogy, curriculum development, and the intersection of computer science and equity, I find myself even more excited about the future of teaching.

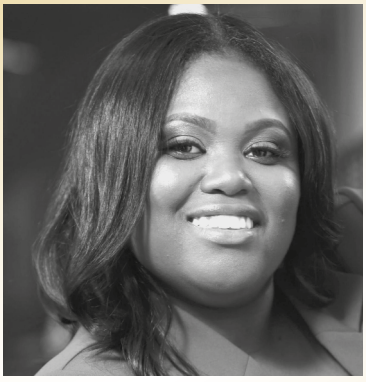
This formal learning is helping me refine my instructional strategies, stay current with trends in CS education, and engage in action research that directly impacts my students. It has also deepened my commitment to building a more inclusive and engaging programming classroom.

Supporting other teachers

The more I grew through CPD, the more I felt called to support others. I began leading CPD sessions for non-CS teachers who were looking to integrate STEM into their instruction. Many of them shared the same feelings I once had — uncertainty, fear of failure, or the belief that 'I'm not a computer person'. Because I've walked that path myself, I can speak to them with empathy, share practical entry points, and help demystify programming concepts.

Teaching programming isn't easy, but it's worth it. And continuous professional development has been my bridge from uncertainty to confidence. Every bootcamp, every workshop, every late-night curriculum tweak was a step towards becoming the teacher my students deserve.

Now, when a colleague tells me, "I could never teach that," I smile and say, "I used to think the same thing. Let me show you how I got started." (HW)



LEONTAE GRAY WARD

Leontae is a STEM and computer science educator dedicated to expanding access for underrepresented students. She is a Project Lead The Way instructor for the School City of Hammond district in Indiana, USA. Leontae is pursuing her MEd in computer science education at the University of Maryland, Baltimore County.

A GLIMPSE INTO THE FUTURE

Lauren Kisser discusses Amazon Future Engineer UK and how educators can help their students prepare for the future

In an age of continuous technological advancement, the landscape of future careers is constantly shifting. It is vital that we show students that there are a diverse range of tech careers and that we must all learn to work creatively with emerging technologies. We caught up with Lauren Kisser to discuss how Amazon is making tech careers more accessible and understandable for young people, and how teachers can integrate careers guidance into their lessons.

Tell us about your journey in technology and what drives your passion for supporting young people in education.

My journey in tech has been an amazing and unexpected adventure. I began with a passion to solve business problems using technology and saw how tech can bring people together through compelling experiences. This led me to my role at Prime Video. Every day, I work with world-class engineers and technical product managers innovating the entertainment experience on behalf of our global customers.

What really drives my passion for supporting young people is seeing that moment of discovery when a student realises they could have a future in tech. Every time I join conversations or see students engaging with our career-exploration initiatives, I'm reminded of how important early exposure to tech careers is. I didn't have these opportunities growing up, and I often think about how different my journey might have been if I'd been able to virtually step inside a tech company or chat with professionals already working in the field.

As a woman in tech leadership, what would you tell your 16-year-old self about pursuing a career in technology?

I'd have quite the pep talk with her and tell her not to let anyone define what she can or can't do in tech. Back then, I had this narrow view of what a technology career looked like — I thought it was all about coding in a dark room somewhere. Now I know that couldn't be further from the truth.

What's amazing about technology careers today is their incredible diversity — every industry, from agriculture to zoology, utilises technology to solve problems. In my current role, I work with artists, storytellers, engineers, and innovators all using technology in different ways. We're passionate about giving real-world examples of tech-forward careers to students through our programmes with Amazon Future Engineer.

How are emerging technologies like AI and machine learning changing the entertainment industry, and what does this mean for future career opportunities?

They are revolutionising work in ways that would have seemed like science fiction just a few years ago. But what really excites me is how these technologies can create positive social impact. One of my favourite examples is voice-activated technology, which is transforming daily life for people who are blind or partially sighted.

We need young people who can think creatively about how to use these technologies to solve real-world problems. We recently carried out research with Gallup which showed that Amazon hires talent in roles of what we call 'careers of the future' — high-growth, automation-resistant jobs that will be crucial in the coming decades (**helloworld.cc/gallup-future**). We're talking about roles like solutions architects, AI specialists, data scientists, and product and

programme managers. Through our educational programmes, we're showing students that tech careers aren't just about writing code — they're about building technology to make life better for people.

How can we make tech careers more accessible and understandable for young people?

Some other research we completed with Gallup revealed that many young people simply aren't aware of the diverse range of careers available in technology (**helloworld.cc/afe-careers**). You can't aspire to what you can't see. That's why we've created multiple entry points for career exploration, meeting students wherever they are in their journey.

For example, through our Career Tours programme, students can virtually experience different parts of the Amazon business, from working in a fulfilment centre, to designing cloud computing solutions, to delivering world-class video experiences (**helloworld.cc/afe-career-tours**). Then we have our Class Chats programme which offers bite-sized, scenario-based learning directly from professionals (**helloworld.cc/afe-class-chats**), and our Future Careers Experience for in-depth virtual work experience (**helloworld.cc/afe-career-experience**).

Virtual experiences are vital in creating equitable access to career exploration, particularly for students from underserved communities. When physical visits aren't possible due to location or resources, technology brings these opportunities directly to students.

> "HIGH-GROWTH, AUTOMATION-RESISTANT JOBS WILL BE CRUCIAL IN THE COMING DECADES

What advice would you give to teachers who want to help their students explore careers in technology and innovation?

The path into tech isn't one-size-fits-all anymore. Whether through university, degree apprenticeships, or career changes later in life, there are multiple routes to success. It's powerful when students realise there's a path that could work for them — it opens up possibilities they might not have considered before.

Representation is crucial in all careers, particularly technology careers, to ensure we are building products that work for all customers. When students see professionals who look like them, who come from similar backgrounds, or who have followed non-traditional paths, it helps challenge stereotypes about who belongs in tech. In our virtual experiences and career programmes, we deliberately showcase diverse role models across different roles and levels. This representation helps students, particularly those from underrepresented groups, envision themselves in these careers.

What's also important is making these experiences relevant to students' lives and interests. When students see how technology connects to things they care about — whether that's entertainment, sustainability, or solving community challenges — they're more likely to engage. The key is showing them that careers in technology are about solving problems and creating solutions that make a difference in people's lives.

How can educators incorporate career guidance programmes into their teaching?

Our programmes are designed to complement existing curriculum and career guidance activities. Teachers can start with the short Career Tours to introduce different aspects of technology, then use the Class Chats for more focused exploration, and finally offer the Future Careers Experience to students wanting to dive deeper. All of these programmes are aligned with Gatsby Benchmarks too [evidence-based benchmarks for career guidance for young people, used in secondary schools and colleges across England; **gatsbybenchmarks.org.uk**].

Beyond traditional pathways, we're also deeply committed to apprenticeships as a vital route into tech careers. In 2025, Amazon announced 1,000 new apprenticeship roles across the UK, demonstrating our continued investment in creating diverse pathways into the technology sector. These opportunities range from entry-level to degree apprenticeships, offering real-world experience alongside formal training (**amazonapprenticeships.co.uk**).

It's been a pleasure talking to you Lauren! Any final words?

I'm incredibly passionate about the work we're doing with Amazon Future Engineer! By providing free, accessible resources and diverse virtual experiences, we're working to ensure that every student, regardless of their background, has the opportunity to explore and pursue careers in technology. It's about building a more inclusive, innovative future — and I believe educators play a big part in making that vision a reality. Check out our free resources at **amazonfutureengineer.co.uk**. (HW)

LAUREN KISSER

Lauren fosters innovation in entertainment as director of Prime Video at Amazon. She champions education as sponsor of Amazon Future Engineer UK, Amazon's childhood-to-career programme which removes barriers for students from underserved communities in computer science and careers of the future.

FEATURE

© Rawpixel.com/stock.adobe.com

NO SCREENS, NO PROBLEM

Rediscovering the power of physical thinking in a digital world

One of the stickiest myths about computer science is that it has anything to do with computers. After all, you can't spell computer science without 'computer', right? That's what I thought — until I actually had to teach the subject.

After 15 years shipping code as a professional software developer, I equated progress with lines of code and productivity with screen time. So I filled my first weeks of lessons as a new teacher with IDE demos and coding drills, all the while watching my students' eyes glaze over. Recursion, data structures, and even simple loops turned into abstract hurdles instead of usable concepts.

My nerds thrived, but the rest decided that computers probably weren't for them. The breakthrough came the day I pulled the power strips, handed out sticky notes, and turned the class into a living neural network. Watching thirty teenagers shout activations across the room and then vote, revise, and cheer at their final classifications taught me that the quickest route to comprehension is often through cardboard, sneakers, and a hallway — not a glowing screen.

Why bodies beat bytes

Before we jump into how I implement unplugged computing activities in the classroom, let's explore why physical learning creates such powerful cognitive anchors.

When students use their bodies to model computational concepts, they engage multiple sensory systems at once, creating richer neural connections than screen-based learning alone. This concept is called embodied cognition, which is the idea that thought isn't confined to the brain but extends to our physical interactions with the world. In other words, by physically acting out abstract concepts, students' comprehension and retention improve dramatically.

The combination of movement, social interaction, and tangible manipulatives creates what cognitive scientists call 'multimodal' learning experiences. More practically, though, physical tasks break screen fatigue. By engaging students in the real world, we give them an opportunity to recharge their mental batteries, creating a natural break that can revitalise focus and renew engagement.

Three off-screen demos that work

By the end of my first year, I had run dozens of unplugged activities, each of which taught my students more deeply than my rambling ever could. But three in particular proved so versatile and impactful that they have become staples in my teaching toolkit, each addressing different computational concepts while engaging students physically and socially.

Living neural net

This unplugged activity transforms students into the individual neurons of a neural network, physically demonstrating how machine learning systems classify data through weight adjustments and activation functions.

To execute it, have students arrange themselves in layers (input, hidden, output), passing sticky notes with 'activation values' between them according to simple rules. For example, in our human neural net, we could classify if a photograph is of a sandwich or not.

Students form the input layer and receive features from the 'image' (like bread type, fillings, condiments). The hidden layer neurons might apply weights to different combinations (bread + meat might strongly suggest 'sandwich' while bread + ice cream would not). The output layer makes the final determination based on the weighted inputs it receives, determining the classification: sandwich or not-sandwich.

After each classification attempt, they adjust their 'weights' (written on their desks or lanyards) until the network successfully identifies patterns.

What makes this activity powerful is how it demystifies the 'black box' of AI. Students viscerally understand backpropagation by physically walking backwards through the network to adjust their values — something that remains abstract when viewed only as matrix maths on a screen.

Markov paper chain

Markov chains might sound like an arcane statistical concept, but they're the backbone of everything from predictive text to route planning (**helloworld.cc/markov-llm**). To demonstrate how past states influence future probabilities, I created a simple exercise using nothing but library books.

The exercise begins with students pairing up with a favourite text (a novel, poem, or essay) to extract 'states' (words or phrases) and 'transitions' (what follows those words). Each pair creates a simple Markov model by scanning through books, recording every instance of keywords and noting what follows each one (for example, if they're tracking the word 'through', they might find it's followed by 'the' 40 percent of the time, 'a' 30 percent, and 'all' 30 percent).

After their data collection, students assemble these probability distributions into a physical chain on the whiteboard using different coloured sticky notes for the 'states' and arrows for 'transitions'. The class then collectively 'walks' the chains, making random choices according to the recorded probabilities at each junction, generating sentences that (loosely) mimic the original source material but with unique variations.

What's remarkable is how quickly students grasp concepts like randomisation, probability distributions, and generative processes through this hands-on exercise, all without writing a single line of code. The tactile nature of sorting sticky notes and physically moving through decision points creates an intuitive understanding that typing into a terminal simply cannot match.

DNS race

Whether you are teaching networking or not, understanding the Domain Name System (DNS) is a critical skill for students to navigate our increasingly connected world, as it forms the bedrock of web browsing, API calls, and nearly every internet-based interaction. This relay race brings the hierarchical lookup process to life through physical movement. To get started, all you need are notecards, masking tape, a long hallway, and a very forgiving school administrator.

In this activity, students become components of the DNS hierarchy. I designate specific students as root servers, top-level domain servers (.com, .org, etc.), and authoritative nameservers for popular domains. The remaining students act as clients, making DNS requests. When a client needs to resolve a domain like 'youtube.com', they must physically run to the correct sequence of servers, collecting pieces of the IP address along the way. The fastest to complete the DNS resolution and return to their starting point (with the right IP address) wins!

To make it even more realistic, you can introduce network latency by requiring students to hop on one foot when they reach 'overloaded' servers, and incorporate caching by allowing students to shortcut the lookup process after their first successful resolution.

What makes this exercise particularly effective is how it physically demonstrates the distributed nature of DNS. Students quite literally experience the hierarchical lookup process, understanding why DNS resolution sometimes fails and how caching might improve performance. Ultimately, this gives them context that purely technical explanations often fail to provide.

> "UNPLUGGED ACTIVITIES CAN SAVE HOURS OF DEBUGGING CONFUSION LATER, AS STUDENTS DEVELOP STRONGER MENTAL MODELS

ZACHARY FLOWER

Zachary is a former software developer turned high-school computer science teacher. After 15 years in industry, he now channels his industry experience into engaging, hands-on projects that make abstract computing concepts tangible and career-relevant **(linkedin.com/in/zachflower)**.

Unplugged by default

After a year of experimentation, I've come to the conclusion that unplugged activities shouldn't be treated as a backup plan for when the Wi-Fi goes down. They should be deliberately integrated into the curriculum as essential pedagogical tools. Spending 30 minutes on any bodied activity can save hours of debugging confusion later, as students approach coding tasks with stronger mental models of how algorithms and other computing concepts work. Thinking computationally has never required a computer — even the earliest algorithms were designed with clay tablets and abacuses — and what our students need isn't more screen time, but a deeper conceptual foundation that transcends any particular technology.

For educators looking to incorporate unplugged computing into their classrooms, all you have to do is start small. Replace one lecture with a comparable physical activity and observe how comprehension changes. After all, the most valuable gift we can give our students isn't mastery of any particular programming language or framework, but the ability to think computationally in any context, screen-based or not. (HW)

THE 'REEL' PROBLEM IN CLASSROOMS

How short-form video apps are reshaping student attention, behaviour, and engagement in computing classrooms

It starts with a swipe. TikTok, Instagram Reels, YouTube Shorts — each is engineered to keep users engaged for as long as possible. The content is short, sharp, and algorithmically targeted, offering an instant dopamine hit with minimal cognitive effort.

It is addictive by design. The type of content our learners consume is a discussion for another article. This is not about blaming TikTok and others; it is about recognising the design patterns of the platforms our students inhabit, and how those patterns shape their expectations, behaviour, and even their capacity for problem-solving.

The fast information, swipe-to-ignore culture, and low tolerance for anything not 'for you' have a clear classroom impact. In computing, where students often work in front of a screen, the conditioning is obvious. They crave stimulation, respond less patiently to failure, and expect instant feedback. Tasks requiring sustained focus, like debugging code or designing algorithms, can quickly lead to frustration or disengagement.

Brains wired for dopamine

Social media platforms run on reward loops: you scroll, the algorithm serves something you will probably like, and you get a dopamine hit. This loop is fast, constant, and satisfying, which is why it is so difficult to stop.

For adolescents, whose executive function and impulse control are still developing, the effect is amplified. When we ask students to debug a five-line Python program, there is none of the instant novelty they are accustomed to. This is not laziness — it is a mismatch between the reward systems they know and the type of cognitive effort we ask for. It also explains why some click wildly between tabs, give up at the first syntax error, or constantly seek reassurance. They have been conditioned to scroll through failure, not sit with it.

Reels vs. reality

In computing, patience is a core skill. Whether writing an algorithm, planning a solution, or evaluating work, success comes through iteration and reflection. But if a student's digital life teaches them that every three seconds something new will happen, how do we help them slow down?

Rather than fight these instincts, I have learnt to adapt. Short attention spans are not a deficit, they are a clue. If students are used to learning through fast, visual content, I need to meet them partway.

For example, in one unit of work, students dissect the algorithms behind the 'For You Page' (FYP): how data is collected, how recommendations are made, and what trade-offs exist. It is a natural entry point into algorithmic thinking, bias, and ethics. They are hooked — not because it is TikTok, but because they finally understand the hidden systems behind the apps they use daily.

I have also introduced micro-challenges — quick five-minute tasks with a clear output, such as writing a snippet of pseudocode, correcting an error, or designing a logical process. These mimic the quick-feedback nature of scrolling while building stamina for longer projects. One of my favourite moments came when a Key Stage 3 (11- to 14-year-old) student designed a flowchart simulating a TikTok-style algorithm. He proudly explained how his 'watch time' metric adjusted content delivery based on interaction. For him, computing suddenly became relevant and creative, and the confidence boost carried over into later coding tasks.

Scroll with purpose

We can also help students become more critical of the media they consume. In Key Stage 4 digital literacy (ages 14–16), I ask, "What makes a video 'scroll-stopping'?"

TRY THIS

Use TikTok's recommendation algorithm as a starter task. How might it work? What data does it use? Then ask students to design their own 'For You' feed algorithm using flowcharts or pseudocode. It is an engaging way to explore algorithms, logic, and ethics in a context they already understand.

Young people are growing up in a swipe-to-ignore culture

© Xavier Lorenzo/stock.adobe.com

> THE AIM IS NOT TO BAN THE SCROLL, BUT TO TEACH WHEN TO SCROLL, WHY TO STOP, AND HOW TO THINK DEEPLY WHEN IT MATTERS

Students analyse structure, pacing, and hooks, then plan their own video with a message, from online safety to digital well-being. Designing the content helps them see behind the curtain.

I also adapt how students evidence learning. Flip (**itsflip.com**) allows short verbal reflections to videos or code explanations. OneNote Class Notebook (**onenote.com/classnotebook**) lets them collect screenshots and annotate their learning journey. These mirror their digital habits but turn them into tools for metacognition.

There is also value in linking these skills to career pathways. Analysing social media algorithms naturally leads into discussions on data science, UX design, and AI ethics, showing students that their habits can be a springboard to future opportunities. The aim is not to ban the scroll, but to teach when to scroll, why to stop, and how to think deeply when it matters.

A culture shift

If we want computing to thrive, we need to see digital behaviour as part of our teaching landscape. I have had more success when I understand students' online world, speak their language, and show interest in their tech habits. I have even dabbled in Reels and TikTok's FYP myself. It felt like a scientific experiment, because I did not enjoy it. I noticed shifts in mood and energy, which helped me empathise with students who live in these loops daily. The difference is, I could choose to delete the apps; they often cannot, or will not.

This perspective supports behaviour management too. A disengaged student is not always disruptive; they might simply be struggling with delayed gratification. A calm, structured environment with achievable steps and visible success can help reframe their expectations. We are, in many ways, unteaching the 'instant win' mentality. But we can do it creatively, by weaving computing principles into their world rather than dragging them out of it. If we meet students where they are, scrolling thumbs and all, we can help them see that the systems they use daily are not just entertainment, but gateways into deeper understanding and future careers. (HW)

ROSS BARRETT

Ross is director of IT at Chantry Academy (the Active Learning Trust) in Ipswich, England. He is passionate about digital literacy, curriculum design, and tackling the challenges of tech-native learners.

CODE EDITOR FOR EDUCATION

Two teachers tell us about how they use our text-based coding environment designed for young learners

Technology in education can be overwhelming, for both teachers and students. With so many tools available, it's hard to know what fits the classroom. A platform that works in a professional or informal setting isn't necessarily going to be suitable in an educational one. That's why we built the Code Editor for Education (helloworld.cc/code-editor-for-ed) in response to the lack of suitable classroom tools for teaching text-based coding.

Currently in beta, our platform is being actively shaped by real teacher feedback and aligned with emerging technologies to ensure it meets the evolving needs of today's classrooms. All features, both current and in the pipeline, are developed with teachers' needs front and centre. It's accessed in your web browser, and getting started is easy. Add an unlimited number of teachers, students, and projects to your verified school account, all for free, forever.

Jeremy Hieb
Integrated maths teacher

Hi Jeremy. Can you tell us a little about yourself?
I teach sixth- and eighth-grade students (11- to 14-year-olds) at a suburban public school north of the Twin Cities Metro in Minnesota, USA. My focus for sixth-graders is digital manufacturing, including 3D printing and laser cutting. For eighth-graders, I primarily teach coding.

What is the primary goal of your eighth-grade coding class?
Students usually arrive with some block-based coding experience from activities earlier on in school. The main objective for my eighth-grade class, set by the school, is to transition them from block-based to text-based coding systems. I chose Python for this transition about eight years ago, and I've been fortunate because of the abundance of resources and its continued relevance.

JEREMY HIEB
Jeremy's love for teaching began when he taught in a rural village school in Tanzania. Since earning his master's in teaching mathematics, he's been teaching integrated maths in middle school in Saint Paul, Minnesota, USA.

How did you personally learn Python to teach these classes?
My learning journey involved a lot of self-study, primarily through the Raspberry Pi Foundation's learning section and by reading multiple basic Python books. My first year of teaching was a steep learning curve with a lot of late nights, as I was still solidifying my own understanding of the curriculum.

What are some common misconceptions about coding you encounter with students?
That coding is only for smart kids. A big part of my job is to convince students that they are capable of coding and can succeed.

What challenges have you faced with integrated development environments (IDEs) for classroom use?
We've been actively searching for a suitable coding platform. We previously used Replit, but it became problematic when they introduced AI features that auto-completed student code. They eventually removed their educational programme, so we had to find alternatives. My hope is that Code Editor will be a reliable, consistent, and fast browser-based IDE that students can use both at school and at home on their Chromebooks.

What feedback do you have on Code Editor, and what features do you find beneficial or lacking?
Code Editor is beneficial because it allows for well-organised programs that students can modify, which saves time and lets us focus on specific skills. However, as the product continues to be developed, there are areas I'd suggest for improvement, such as adding simultaneous coding, which I find valuable for collaborative group assignments and fostering social interaction among students.

What is your biggest hope for your students?
My biggest hope is that my students become effective problem-solvers. I view programming as a powerful tool to develop critical thinking and creativity, and I want them to embrace that.

What advice would you give to teachers who are hesitant to start teaching computer science?
My advice is: don't be afraid to fail. Teaching, especially in a rapidly evolving field like computer science, is an iterative process. It's important to admit when something isn't working and to continuously modify your approaches until they are effective for most students.

Tom Mason
Head of mathematics and ICT

Hi Tom. Tell us a little about yourself.
I'm a teacher in Croydon, southeast London, in England. I've been head of maths for a couple of years and am now also head of computing.

Did you have a background in computer science before you started teaching it?
I was a maths teacher, but during my undergraduate degree I did some computing modules and a little bit of robotics, so I've always been interested in computing. About four years ago, the school asked if I'd like to teach IT and computer science, and I said yes. I've been teaching A-level (16- to 18-year-olds) and GCSE computer science (15- to 16-year-olds) since then.

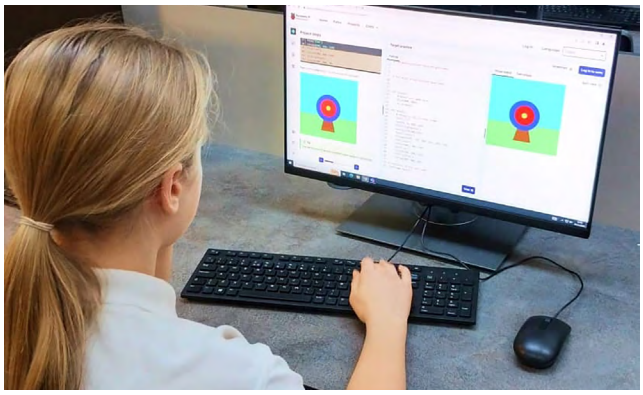
What is your classroom like?
It's an average-sized, all-boys Catholic school in a lower socioeconomic area. We don't have a big budget, and many students have limited access to computers at home, so some do their coding on their phones. It's crucial for us to teach computing well in school so they don't miss out.

How did you first learn about the Code Editor?
I was looking for an in-browser IDE because we can't install Python natively on school computers for safety reasons. I tried Trinket and Replit, but then I found the Raspberry Pi Foundation's Code Editor through a Google search. The fact that I could create classes and projects, and that it was free (unlike Replit, which started charging), ticked all the boxes.

What were your first impressions of Code Editor?
I started using it just before Christmas. It was a little clunky to set up at first, as I had to add students one by one (now you can import from a spreadsheet). But as soon as students started using it, it was brilliant. I can send projects to all students at once, and they each have their own editable version, similar to Google Docs. The addition of markdown, which allows me to embed lessons and instructions, was a huge improvement. It also supports HTML and CSS, which I've used for my A-level classes.

Can you share an example of a particularly successful lesson?
My Year 10 class, who are now going into Year 11, created a ten-question quiz about Python. I gave them minimal direction, encouraging them to think outside the box, randomise questions, and even hide answers in separate files. They came up with incredibly clever and creative solutions, and it was clear they were genuinely understanding the code. This open-ended, project-based learning worked really well because they couldn't just google the answer; they had to explore and problem-solve.

Why should other educators try Code Editor?
It's a very simple tool that does almost everything a teacher needs right now, and new features are constantly being added. Most importantly, it doesn't have AI auto-completion that inhibits students' learning. It's easy to set up, works really well, and I can see all my students' work live. I think it's brilliant.



Code Editor for Education has classroom features for educators

What is your hope for the students you're teaching to code today?
I'd love for them to be developing code for others, whether in industry or at university, and to be changing the world. I want them to be using fresh, new technologies and coming up with innovative ways to use AI that I could never imagine.

What do you find is the biggest challenge in teaching computer science right now, especially with the rise of AI?
The biggest challenge is students cheating by using AI to write code. While AI is used by professional developers, I want my students to learn to code themselves. It's a catch-22. I try to encourage them to use their brains first, and if they're stuck, to ask specific questions of AI tools such as, 'How do I swap two entries in a list?' rather than 'Tell me how to do bubble sort in Python code.' This helps them understand the code, instead of just copying and pasting.

What advice would you give to a teacher hesitant to start teaching computer science?
The technology is really cool! It's fun to tinker with things. Look at some code, try to break it, and figure out why it can't do certain things. Push the code to its limits. This will help you understand how it works.

Thank you, Jeremy and Tom. If you'd like to try Code Editor for Education, visit helloworld.cc/code-editor-for-ed. (HW)

TOM MASON

Tom is head of mathematics and head of ICT at an all-boys secondary school in southeast London. He is passionate about teaching and learning, and has a keen interest in digital education practices.

RAISING CODERS IN THE AGE OF AI

Ryan Etheridge shares how his family uses AI as a tool for coding, problem-solving, and meaningful projects in rural North Carolina in the United States

I'm a dad of three boys growing up in a one-stoplight town in the foothills of western North Carolina, USA. Out here, there's no shortage of fresh air or room to run, but finding activities that excite a kid's mind can take some work! What we do have are front porches, long car rides, and plenty of evenings around the coffee table. That's where my three boys (ages 13, 11, and 8) and I have found space to build a practice of coding and, more recently, artificial intelligence.

At first, I worried. Would AI wipe away all the hard work we'd put into learning the basics of coding? Would loops and logic go the way of the rotary phone? But I came to see something different: AI wasn't a replacement for thinking, it was a tool that could help my boys think bigger, ask sharper questions, and create with more imagination than before.

Goals with a purpose

We start every summer by setting goals, not just for chores or sports, but for coding. This year, we used an AI tool to set weekly construction and coding goals to be used with a LEGO® Spike™ kit the kids got as a Christmas gift from grandparents last year. The boys shared their interests — outdoor pursuits, soccer, board games, Nintendo Switch, Beyblade X, and reading — which the tool turned into eight weekly lessons like 'Robo-Goalie Showdown: Build and code a robotic goalie that can block a ball using sensors and timed reactions.' They ended up focusing completely on shooting and made the robot shoot the ball at varying strengths based on how hard the force sensor was pressed. It was awesome!

We began to incorporate AI into other domains of summer decision-making. In early May, I realised we had a free week and needed to plan a vacation. Sitting together, I told the boys: First, let's define the goal: Will you help me plan a summer vacation for my family? Then I added the role: I want you to serve as my trip planner and travel agent. Finally, I invited my kids back into the conversation: Ask us questions to shape the trip in a way that aligns with our interests and means.

That three-step rhythm — goal, role, clarifier — is the simplified version of the prompt-engineering training I lead with the staff at my school. For kids, introducing AI isn't about typing fancy commands but about thinking clearly, defining purpose, and setting boundaries. The follow-up questions from the AI tool helped the boys see how constraints (like budget or time) could actually improve results. I had a good time making up counterexamples on the fly to show what would've happened if we said

The weather exhibit at the North Carolina Museum of Natural Sciences appeals to the boys' love of science and nature

PROMPTING MADE SIMPLE

Try our family's three-step method on your next car ride or long walk:

1. Define the goal.
2. Assign a role to the AI tool.
3. Add a clarifier to keep the human conversation going.

This works for coding, trip planning, or any project where kids need both structure and creativity.

we loved lazy rivers instead of rollercoasters.

At that moment, coding wasn't a worksheet or tutorial. It was real. It mattered to them. And it set the tone for a summer where AI wasn't a machine doing the thinking for us, but a neighbour lending a hand while we steered the work.

Car conversations

Our best AI conversations don't happen at a desk. They happen in the car. Something about the rhythm of the road gets my boys asking questions: Could AI design an album cover for our band, *The Meaty Ogres* (a play on 'mediocre' because I'm such a terrible singer)? Could it help generate scavenger hunt clues for a friend's birthday? When my wife drives, I type their questions in while we discuss the goal, the role, and the clarifiers. Then we read the answers out loud and argue, laugh, or poke holes in what the AI says.

These rolling conversations have become a family classroom. They teach the boys that AI isn't always right. Sometimes it gets the details wrong, or gives answers that just don't fit. That opens the door to critical thinking: How do you know if something's true? What sources matter? How can you test it yourself? In the full AI training for my staff, the final guiding point is 'Keep the human in the loop.' It's the same lesson I want for my sons. The real joy of the work comes from staying engaged, asking questions, spotting errors, and shaping the outcome ourselves.

Living rurally, with fewer formal programmes, we've learnt that the family minivan can double as a lab. It's where prompt engineering turns into road-trip laughter and where critical thinking is woven into long miles.

Coding meets life

I've always told my sons: you don't learn skills in isolation. You learn them when there's something meaningful at stake. So when we wanted to build a movement tracker, we didn't just ask, "How do we code it?" Instead, we asked, "How do we make it fun enough that we'll actually use it?"

AI has become a companion in these projects. It helps generate ideas, troubleshoot problems, and sometimes explain maths concepts in new ways. When my middle son wrestled with algebra, we used AI tools to visualise equations and explain the concepts using practical, hands-on examples. When my youngest worried that AI might kill off polar bears because of its energy use, we turned that concern into an opportunity to research sustainable computing practices and talk about climate responsibility.

> "AI IS LIKE A NEIGHBOUR LENDING A TOOL, BUT YOU NEED TO KNOW WHAT YOU'RE BUILDING

The beauty of it is that AI always bends back towards the human work of deciding what matters. My boys are learning that coding isn't about writing perfect lines or arranging the blocks the right way but about solving problems that touch their lives and their values.

Neighbours, not replacements

In our little community, we know the value of good neighbours. AI, to me, is like a neighbour willing to lend a tool, a helping hand, or a cup of sugar. But you still need to know what you're building. That's the lesson I want my kids to carry with them: AI is powerful, but only when you set the purpose, define the role, and stay in the driver's seat.

For rural families like ours, coding with an AI helper opens doors that otherwise might stay shut. It connects kids to global communities, gives them a playground for ideas, and prepares them for a world where technology touches every field. But at its heart, it's still about raising kids who can think, dream, and create with their own hands and minds.

That's why I believe programming in the age of AI is more important than ever — not because AI will do the work for us, but because it challenges us to become better question-askers, sharper thinkers, and more purposeful makers. Out here in our one-stoplight town, that feels like a future worth building. (HW)

The three boys love heading into the city to visit museums — whether we're at home, in the car, or enjoying an illusion exhibit in Charlotte, the Etheridge boys have opportunities to think, dream, and create

RYAN ETHERIDGE

Ryan is an elementary school principal in rural North Carolina, USA. He explores mathematics, coding, and AI with his three sons, blending hands-on learning with semi-structured play.

BRIDGING THE GAP

Using semantic waves to help scaffold problem-based learning in computer science education

Problem-based learning (PBL) is a popular approach in computer science education. Its appeal lies in the belief that students learn coding best by tackling real-world problems. The hope is that through authentic engagement, learners will naturally acquire computational thinking skills. In practice, however, this approach demands far more of teachers than is often acknowledged. Without careful scaffolding, the gap between students' everyday understanding and the abstract thinking that is required in computing can be overwhelming.

The unique challenge of computer science

Teaching any subject involves navigating abstraction, whether it's glaciation in geography, nationalism in history, or irony in English. Teachers rely on metaphors, examples, and lived experience to connect ideas to students' existing knowledge. In turn, students are expected to do the reverse: to take personal, concrete experiences and generalise them into more abstract, academic language.

This movement between the concrete and the abstract is called a semantic wave, a concept from educational research that describes how meaning is built and transferred through shifts in context and complexity. Effective teaching depends on helping students move fluidly along this wave. But in computer science, the wave is steeper.

The subject combines abstract theoretical concepts with practical implementation. Students must understand code structures like 'for' loops or global variables, but must also learn to apply logic and reasoning to problems drawn from everyday life. Making this leap, particularly from a real-world issue to a digital solution, is often where students struggle most.

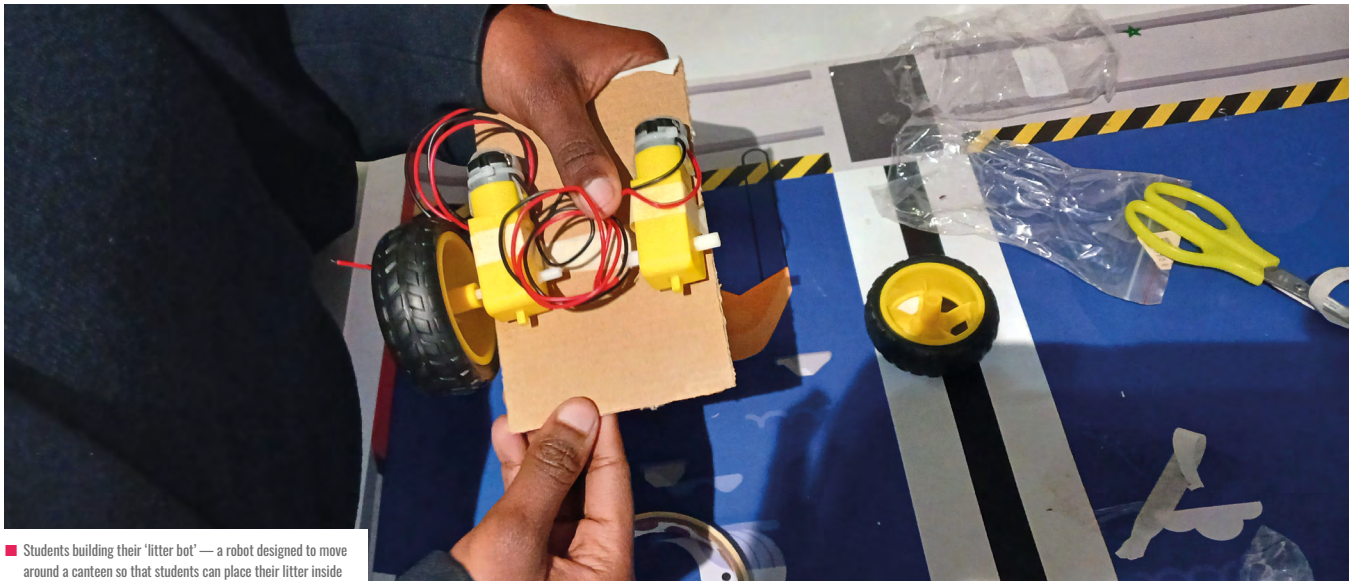
More than just learning to code

Computer science education often assumes that if students understand the syntax of a programming language, they can solve problems. But writing code is only part of the journey. Real learning happens when students use computational tools to model and solve real problems.

For example, students may be asked to code a micro:bit to trigger an alarm when it detects motion. With enough guidance, they'll succeed. However, if the task is more open-ended, say creating a device to address an environmental issue, they often get stuck, not because they can't code, but because they don't know where to begin.

A real-life example

In a recent grade eight (ages 13–14) project, my students were asked to design a prototype to solve a pressing environmental challenge. One student chose forest fires. Her idea was to use a micro:bit chip to



Students building their 'litter bot' — a robot designed to move around a canteen so that students can place their litter inside

detect a fire and send a signal to alert authorities. This was a great start, but she was soon stuck. She couldn't understand how a micro:bit could detect a fire.

I tried to help her break it down. Fires produce heat and light. Sensors can detect both. To simplify, we dropped smoke detection and focused on heat and light levels. Even then, she struggled. The real problem was not the coding, it was thinking through how to define 'fire' in logical, programmable terms. This formed her initial challenge.

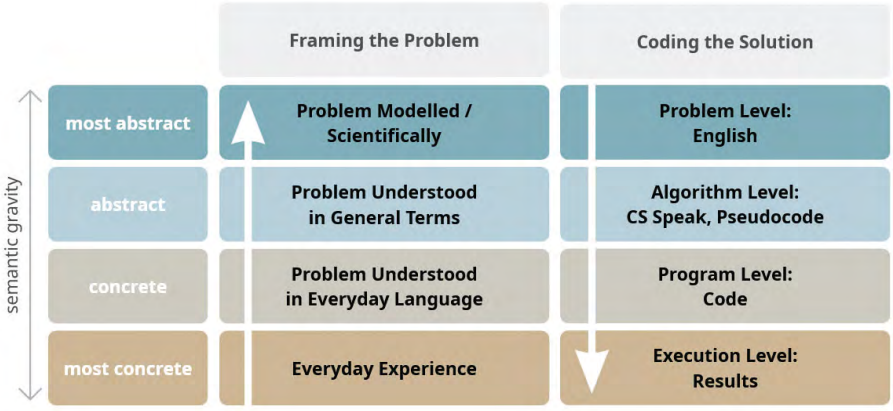


Figure 1 The semantic wave up and down the plane for framing the problem and coding the solution

> MAKING THE LEAP FROM A REAL-WORLD ISSUE TO A DIGITAL SOLUTION IS OFTEN WHERE STUDENTS STRUGGLE THE MOST

I talked her through a possible approach: if both light and temperature exceed a certain threshold, an alarm is triggered. To my mind this seemed a really simple step. It's simple in theory, but if you think about it, it requires multiple conceptual steps: defining the problem, abstracting it into measurable variables, understanding the sensor inputs, and writing the code.

In these struggles, she was not alone. Students working on projects like detecting floods, tracking animals in the wild, or automating recycling bins all encountered similar roadblocks. The challenge was not really the coding itself. It was translating a concrete issue into a generalisable model that could be implemented with hardware and code.

Two layers of complexity

This experience made it clear to me that students face two distinct challenges. First, they must frame the problem. This involves starting from a familiar context, describing it in everyday language, and narrowing it into an abstract, generalisable form.

Then, they must design and implement a solution; moving from the abstract back to the concrete, writing an algorithm, coding it, testing it, and debugging it.

These steps describe two distinct movements up and down the semantic plane (**Figure 1**). Both steps require different cognitive skills and different kinds of support. Without explicit scaffolding, students can quickly lose confidence.

When hands-on learning gets too abstract

PBL and physical computing are meant to make learning more accessible. By involving sensors, motors, and microcontrollers, these methods are supposed to connect theory and practice. Ironically, they often introduce even more layers of abstraction. Now, students must also understand how hardware works and how input from the real world translates into digital logic.

What looks like a smooth path, from real-world problem to coded solution, requires multiple cognitive leaps in reality. Students must understand the issue, model it scientifically, grasp how sensors and devices operate, apply programming structures, and integrate all of this into one functioning system.

The role of the teacher

Teaching computer science is not just about teaching programming. It is about teaching students to view the world through a computational lens. They must learn to break down complex issues into logical steps, build models, and design solutions that can be implemented in code. This requires teachers to scaffold not just the technical skills, but the thinking that comes before the code.

Reflecting on my own teaching, I now recognise that the biggest support students need is in framing the problem. Once the problem is well defined, the code often follows more easily. Helping students move from observation to abstraction, and from abstraction to application, is the real heart of effective computing education. (HW)

DORIAN LOVE

Dorian teaches ICT, coding, and robotics at Roedean School in Johannesburg, South Africa. He is researching computer education, including looking at how grade eight and nine students (13- to 15-year-olds) tackle physical computing projects using micro:bits.

EMBRACING THE THRILL OF THE JOURNEY IN CODING

Sethi De Clercq shares his tips and activity ideas to introduce the PRIMM approach to young digital explorers

In the world of adventure, no one would hand an explorer a map with a completed path to X-marks-the-spot and say, 'Congratulations, you've arrived at your destination!' The thrill, the learning, and the real achievement lie in the journey itself: planning and plotting the course, taking the first steps, reading the landscape, and navigating the pitfalls. Reaching the final destination is simply the evidence of the adventure.

Yet in our earliest computing lessons, we sometimes risk doing the equivalent of handing over the completed map. We show a sequence, the robot arrives, and we celebrate. But what about the adventure of getting there? This is where the PRIMM (Predict–Run–Investigate–Modify–Make) framework comes in (**helloworld.cc/PRIMM**). To some, PRIMM is seen as the domain of secondary-school classrooms grappling with text-based code and complex syntax. But PRIMM is not just for older programmers; it is a fundamental pedagogy for exploring code, perfectly suited to the play-based, discovery-led world of early years foundation stage (EYFS; ages 2–4) and Key Stage 1 (ages 5–7).

Introducing PRIMM

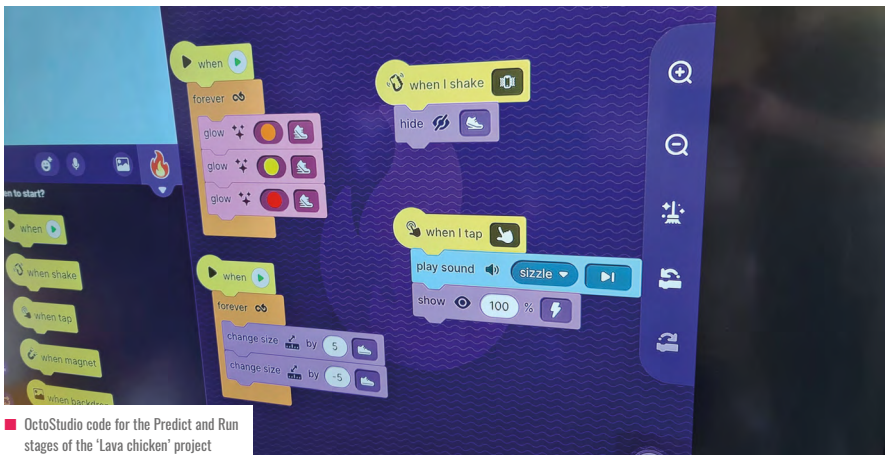
PRIMM provides a five-stage structure for planning programming lessons and activities, which builds understanding layer by layer. It's not about finding the right path immediately; it's about understanding the terrain. As a teacher of young and very young learners, this is what the PRIMM stages generally look like for my digital explorers:

Predict (or 'plotting the course'): this is the 'What do you think will happen?' stage. Before anything moves, before a single block or button is clicked, the child looks at the instructions and makes a hypothesis. It's a low-stakes entry point that values thought over immediate action, and in this fast-paced world full of flashing lights and sounds, it is a very welcome step.

Run (or 'taking the first step'): the child presses 'Go', literally or metaphorically! They run the code, follow the instructions, or watch the sequence play out. This is the moment of truth when their prediction is tested against reality, and represents a moment of instant feedback. Cue lots of animated reactions!

Investigate (or 'reading the landscape'): here lies the heart of PRIMM. This is the 'Why did that happen?' stage. The child looks closely at each individual instruction or block of code, perhaps discovering that 'That arrow made the robot turn' or 'That green block made the character jump.' This is a slow, deliberate dissection of cause and effect, where the logic of the system is uncovered one step at a time. This step-by-step investigation of the sequence is a powerful learning opportunity for younger learners.

Modify (or 'choosing a new path'): now we get to the tinkering, 'What if' stage — 'What if I change this number?' or 'What if I use a different block here?' Children experiment by making small changes to the existing code or sequence and then



OctoStudio code for the Predict and Run stages of the 'Lava chicken' project

Plotting our course to a campsite on a map

running through a mini Predict/Run cycle to see what effect their change had. This is where resilience and a no-fear-of-failure mindset are born, helping younger learners to develop a much deeper understanding of the changes they make.

Make (or 'drawing your own map'): equipped with a complete or partial understanding of how the individual steps work, the child is now ready for their own adventure. They are challenged to create a new sequence or program from scratch to solve a new problem. The learner is no longer just a map reader; they are a cartographer.

PRIMM in the EYFS

In early years, PRIMM is all about sequencing, pattern, and logic in the physical world. It's hands-on, tangible, and predominantly screen-free. Here is an example of PRIMM in practice from my EYFS classroom, where learners work towards creating an algorithm for making a bug hotel in nature. For this activity, teachers need a selection of natural materials (hollow sticks, pine cones, small logs, stones, bamboo canes, etc.) and a premade frame or designated area for a bug hotel.

Predict: the teacher lays out a jumbled collection of materials, or demonstrates placing them on top of one another without considering a structure. They ask questions such as, 'What do you think would happen if we just piled all these things up like this? Would the bugs have good homes?' A child might predict, 'The sticks would just fall out!' or 'The small bugs wouldn't have anywhere to hide if it all collapses.'

Run: children attempt to build a section of the bug hotel following the jumbled placement, demonstrating the instability or inefficiency of the initial arrangement.

Investigate: the teacher guides the learners, prompting them to ask questions such as, 'Why did that fall over?', 'Which materials are best for the bottom?', 'Where would the earwigs or isopods like to live best?', and 'What kind of material would make them happy?' Through tinkering, the children identify that larger, sturdier items are needed for the base, and different bugs prefer different types of shelter. They are investigating the structural dependencies, and may even have some ecological considerations.

Modify: the learners now rearrange and re-select materials, discussing how to create stable sections and varied habitats within the hotel. They might decide to use thicker logs for the base, then hollow bamboo canes, and finally straw and leaves for smaller crevices. They run it again by building a new, more stable section.

Make: learners are then faced with a new challenge: 'Can you make a new bug hotel design using different materials? What order would you put them in so it's strong and offers homes for different creatures?'

The pros of this approach are that it helps develop logical reasoning, sequencing skills, problem-solving, and, for this example, an understanding of natural structures and habitats. It makes abstract computational thinking concepts concrete and relatable through an engaging, nature-based activity.

> REACHING THE FINAL DESTINATION IS SIMPLY EVIDENCE OF THE COMPUTING ADVENTURE

However, it also has its cons. Following these stages can be time-intensive, and requires the teacher to guide the Investigate stage with careful questioning about both the engineering and ecological aspects.

PRIMM in Key Stage 1

By Key Stage 1, children are often using tools like Bee-Bots, programmable toys, or simple block-based apps like ScratchJr or OctoStudio. PRIMM provides the perfect structure for these initial coding experiences. In this example, learners program a chicken sprite to fall into a pit of lava in OctoStudio.

> PRIMM BUILDS RESILIENT, CURIOUS, AND LOGICAL THINKERS FROM THE BEGINNING

Predict: the teacher shows a simple, prebuilt program: **when I tap start** block -> **move 3 motion** block -> **turn 90° clockwise motion** block. They ask learners, 'What do you think the chicken will do when it is tapped? Will it reach the lava pit?' Children might predict, 'It will move right, then turn, but it won't go in the lava.'

Run: tap the chicken sprite and children watch it move and turn according to the code. Learners observe its path and where it ends up relative to the lava.

Investigate: the teacher asks, 'Why didn't the chicken land in the lava? What does the **move 3** block do exactly? What happens if we change the block to **move 1**? And what about the **turn 90° clockwise** block?' Children experiment by isolating individual blocks or short sequences, understanding their precise effect on the chicken's movement and orientation.

Modify: teachers now ask questions such as, 'How can the code be changed so the chicken does go into the lava? Which numbers or movement blocks need to be different? Should it turn more, or less?' Children suggest and make changes, such as changing '3' to '5' for move, or adding a **move down** block. They then run the modified code.

Make: now, teachers can set learners off to create a new program, such as making the chicken walk directly into the lava and making a sizzle sound when it touches it.

The pros of this approach are that OctoStudio's visual, tap-based interface is intuitive for this age group. The PRIMM framework naturally differentiates, allowing all children to participate in the Predict and Run stages, while challenging others with more complex Modify and Make tasks. It also introduces foundational programming concepts like sequencing, parameters, debugging, events, and sounds through motivating and playful interaction.

However, the approach can feel slow for children (and educators!) keen to jump straight to the Make stage, requiring emphasis on the value of the Investigate and Modify process. This is a tricky one, as even teachers get very eager to jump in and add a ton of features, animations, and effects!

Reflections

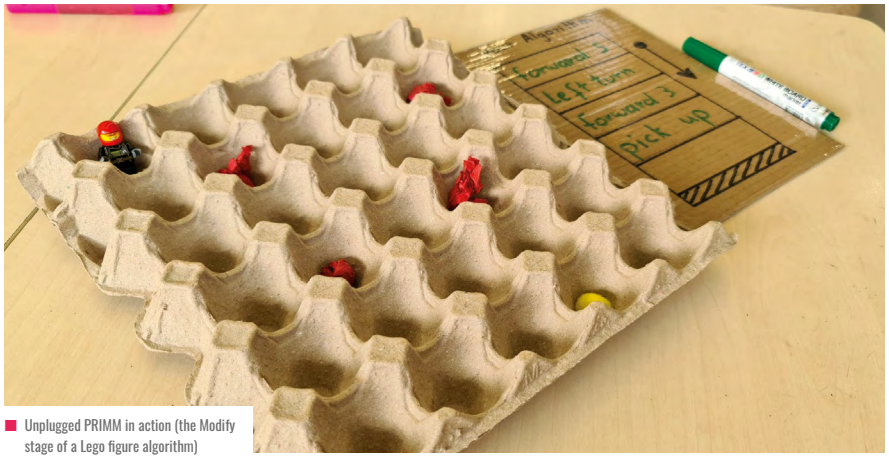
Embracing PRIMM in the early years and at Key Stage 1 is a conscious decision to value process over product. It requires us to resist our instinct to praise only the successful final outcome and instead celebrate the insightful prediction, the careful investigation, and the brave modification. Yes, I do like a good adjective.

The beauty of PRIMM is that it builds resilient, curious, and logical thinkers from the very beginning. When a child's prediction is wrong, it is not a failure; it is the start of a fantastic investigation. When their modification doesn't work, it is not a mistake; it is a new clue they get to explore.

We are not just teaching children to follow instructions or copy code. With PRIMM, we are equipping them to be fearless digital explorers, ready to chart their own course, understand the terrain, and embark on a coding adventure with confidence. Let's not just hand them the completed map; let's teach them the joy of the expedition. (HW)

SETHI DE CLERCQ

Sethi is head of Key Stage 1 and the computing lead at Rugby School Thailand. Passionate about meaningful technology integration across all age groups, he shares training and tutorials on his YouTube channel, and writes on **sethideclercq.com** and **readysetcompute.com**.



Unplugged PRIMM in action (the Modify stage of a Lego figure algorithm)

(Hello World) NEW PODCAST SERIES

Hear the voices behind the stories as we continue the conversations on our podcast:

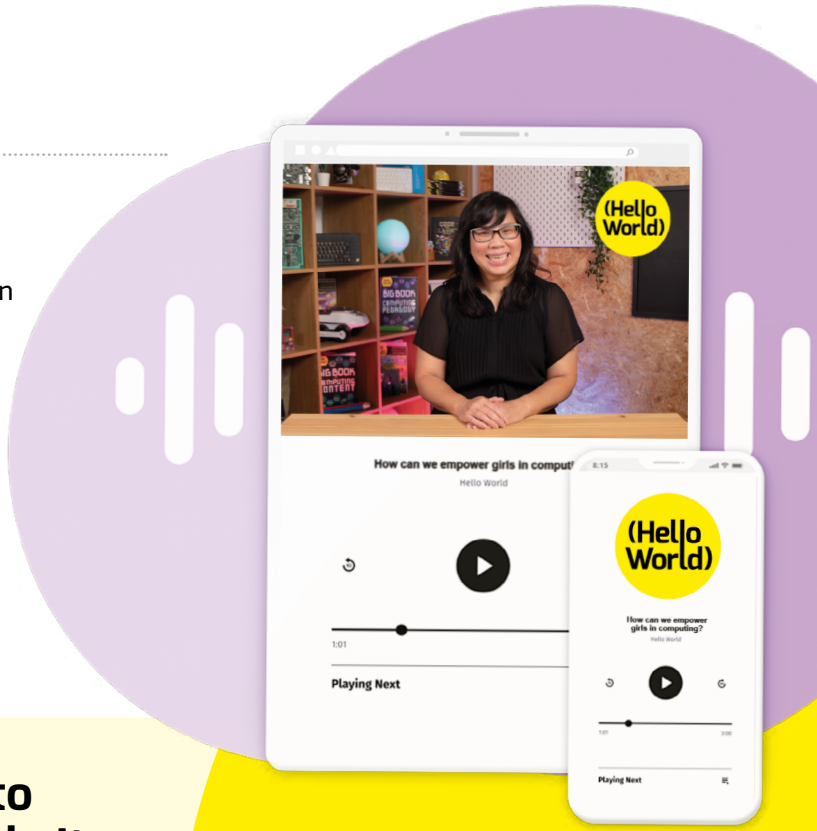
- Listen on your favourite podcast app (search 'Hello World Podcast')
- Or watch them on YouTube

WATCH THEM ON YOUTUBE AND SPOTIFY:

Teacher tips on how to teach programming
helloworld.cc/pod28-tips

"Vibe coding" in programming education? — a research deep dive into "vibe coding" and whether it has a role in programming education
helloworld.cc/pod28-ai

Which programming language should you choose to teach coding? — educators discuss the pros and cons of using Scratch, Python, Java, and other programming languages to teach coding skills at different learning stages **helloworld.cc/pod28-code**



For full episodes and to subscribe, visit our website
helloworld.cc/pod

LESSON PLAN

SEQUENCING

Understand that computers need to be given precise instructions in order to execute them

The following lesson plan is taken from the 'Programming essentials in Scratch: part I' unit from The Computing Curriculum (TCC), written by the Raspberry Pi Foundation. It is aimed at learners aged 11 to 12, to help them understand the precise nature of the instructions that computers need to execute.

ABOUT THE COMPUTING CURRICULUM

Raspberry Pi Foundation

The Computing Curriculum is the Raspberry Pi Foundation's bank of free lesson plans and other resources that offer educators everything they need to teach learners aged 5 to 16. It covers the full breadth of computing, including computing systems, programming, creating media, data and information, and societal impacts of digital technology.

Every unit of work contains a unit overview; a learning graph to show the progression of skills and concepts in a unit; and lesson content, including a lesson plan, slides, and formative assessment opportunities. Find them when you sign up for a free account at **helloworld.cc/tcc**.

AGE RANGE

11–12 years

OBJECTIVES

- Compare how humans and computers understand instructions
- Define a sequence as instructions performed in order, with each executed in turn
- Predict the outcome of a simple sequence
- Modify a sequence

This is the first lesson in The Computing Curriculum's unit 'Programming essentials in Scratch: part I'. Learners will be taught the song *Frère Jacques* before working in pairs to place blocks of code into the appropriate subroutines so that their program will play the song correctly. (HW)

STARTER ACTIVITY: ROBOTS VS. HUMANS

5 MINUTES

Ask learners, 'Can a computer do the job of a musician?' Give them a minute to write a short answer justifying their opinion, then ask them to share it with the person next to them.

Briefly discuss their answers together as a class. Try and steer the conversation towards discussion of the fact that musicians mostly follow sheet music; while a computer could be programmed to do this, ask whether it could add the style, flair, and creativity of a human musician.

REQUIREMENTS

- Access to Scratch 3 either via the web (**scratch.mit.edu**), or offline (**scratch.mit.edu/download**). Check that your school filters do not block any part of this website.
- *Frère Jacques*: version to give learners (**the-cc.io/Y7sequencing1**).
- *Frère Jacques*: completed solution (**the-cc.io/Y7sequencing1-complete**).
- *Frère Jacques*: completed explorer task (**the-cc.io/Y7sequencing1-complete_explorer**).
- You may want learners to have headphones or to be able to control the sound from the speakers, if their devices have them.

ACTIVITY 1: HOW COMPUTERS NEED PRECISE INSTRUCTIONS **10 MINUTES**

Ask for a volunteer from the class. They will play the role of a human-style robot. The rest of the class will take it in turns to play the role of the computer programmer, who is going to give instructions for the robot to follow.

Either draw on the board behind the robot so that they can't see, or give the instructions to the class without the robot hearing, and ask learners to give instructions to the robot, one at a time, for it to carry out the following scenario:

1. Complete a full circuit of the class (depending on the layout of your classroom, you might want to be a little more descriptive to help the learners who are giving the instructions, e.g. tell them where the robot needs to go).

After the robot has carried out the instructions, ask the class to analyse what happened. The following scenarios might have taken place:

- The learner might have followed the instructions like a human rather than a robot, and made up detail that wasn't in the instructions (for example, they knew which direction to turn and how many degrees).
- The learner could have followed the commands to the letter and deliberately walked into objects.
- The learners giving the instructions might have given very precise commands which the robot followed accurately.

Allow the learners to reflect on what happened and what instructions were given to the robot. Discuss with the class the need for precise instructions (for example, the need for more than just 'turn right', adding how far to turn right if needed, i.e. the angle of the turn).

At this point, you might want to give another learner in the class the chance to have a go at being the robot.

2. This time, the computer can use only the following commands:

- Forward (number of footsteps)
- Right (degrees)
- Left (degrees)

Ask the learners to only use these commands in their instructions to move the robot around the classroom. As part of the route you give, make it a secret requirement that they have to go outside the classroom. Make sure the door is shut.

- Ask the class to again reflect on what happened when they got to the door. Did the robot stop?
- Did the robot become a human and 'fill in the gaps' by assuming they needed to open the door to go through it?

Summarise that computers require precise instructions and can only do exactly what they are instructed to do.

Activity 1

The robot and the computer programmer

Volunteer needed

Volunteer 1 will play the role of a human-like robot that can move

The rest of the class will take it in turns to play a computer programmer and will give instructions to the robot

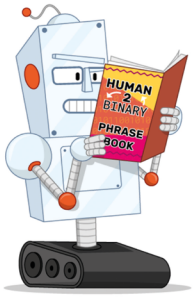


Figure 1

ACTIVITY 2: WHAT IS PROGRAMMING? 5 MINUTES

Programming is how you get computers to solve problems.

Highlight to the learners that as programmers, they will learn how to get a computer to solve problems. Unlike builders, who are constrained by physical limitations such as the number of bricks they have, or the maximum height of a building, as a programmer, there are no such limits. The only thing that limits a programmer is the expanse of their imagination and their ability to use logic and code to solve problems. In this unit, learners will start to become equipped to solve problems using logic and code.

Introduce these key terms:

- Sequence: running instructions in order
- Selection: making choices
- Iteration: doing the same thing more than once

There is no need to go through these words in detail at this stage, only to show that there are different ways to control how your instructions will be executed. Learners will learn more about this during the course of this unit.

Figure 2

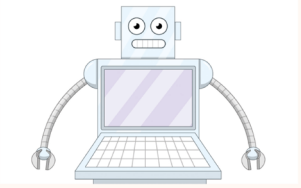
Activity 2

"Programming is how *you* get computers to solve problems"

There are two key phases that are important here:

You: Without the programmer (you), the computer is useless. It does what you tell it to do.

Solve problems: Computers are tools. They are complex tools, admittedly, but they are not mysterious or magical: they exist to automate tasks.



ACTIVITY 3: INTRODUCTION TO SEQUENCING AND MUSIC 5 MINUTES

Define sequencing as running instructions in order. Ask learners to think of any non-computing examples of where instructions need to be carried out in the correct sequence, such as:

- Music scores
- Recipe books
- Furniture assembly instructions (note that Lego characters are also on the image in **Figure 3**; Lego assembly guides are also good examples of sequencing!)

Ask the learners, if a computer was going to be programmed to play music, what extra information would it need?

A computer that was able to output sound would be able to play a sequence, as long as it was programmed to process:

1. What sound each note makes
2. How long to play each note for

Activity 3

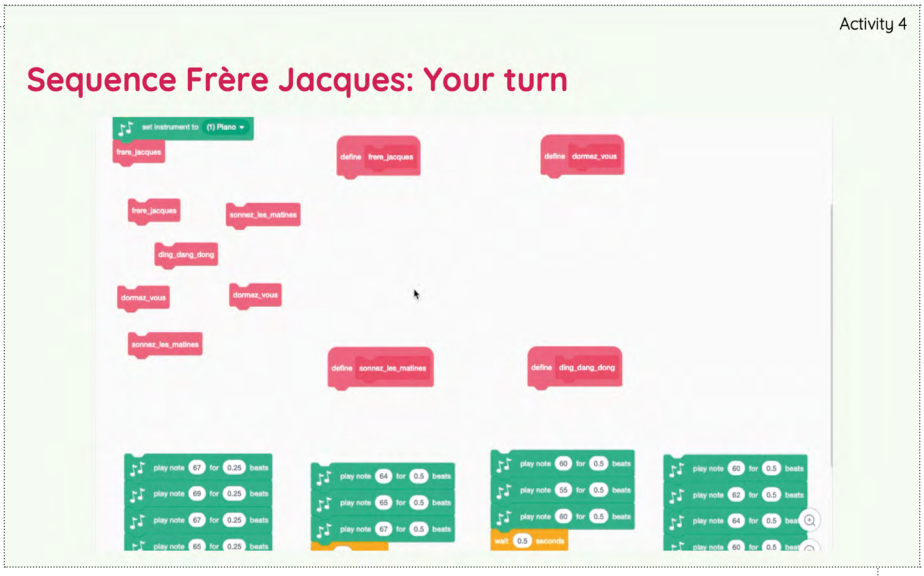
Sequencing: Instructions performed in order, with each executed in turn

As you have seen, computers will follow your instructions precisely and in the order in which you tell it.

Can you think of any non-computing related examples of where instructions need to be carried out in the correct sequence?

Figure 3

Figure 4



ACTIVITY 4: *FRÈRE JACQUES* 20 MINUTES

Introduce the concept of pair programming. Learners will be placed into pairs to complete the work. They will take turns to be the driver and the navigator:

- The driver's role is to control the keyboard and mouse and place the code blocks into the correct places
- The navigator's role is to support the driver by watching for any mistakes, reading the instructions to the driver, and seeking support if needed

Arrange the learners into pairs, explain the task, and distribute the worksheet for learners to complete (**Figure 4**). Normally, when using pair programming you would swap roles every five minutes. As this isn't an overly long activity, we recommend that you instruct the learners to swap roles at the start of each new task.

The aim of this activity is for the learners to use the Scratch program to place the blocks into the correct places to create a working program that will play *Frère Jacques*.

Ask learners to open the *Frère Jacques* (**the-cc.io/Y7sequencing1**) Scratch program.

Task 1: the *Frère Jacques* sequence

- Move the blocks into the correct sequence so that the lyrics to the song are in the correct order.
- To help you, try listening to the melody again (**helloworld.cc/frere-jacques**).

Task 2: make your subroutines

- Listen to each block of music sequences (the green blocks) by using a single click on the block.
- Place the sequences under the appropriate subroutine headings.

Task 3: listen to your music

- Now press on the green flag on the top right-hand side of the screen to listen to your program play.

PLENARY 5–10 MINUTES

This task is designed to reinforce the key concepts from this lesson (sequencing, and the fact that computers need precise instructions).

Print the 'AP-worksheet' (**helloworld.cc/sequencing**) or write the following words on cards: Boing, Bang, Wow, Splat, Kerpow, and Boom. Add two image cards, for example a picture of a lion and a picture of a splash.

Ask eight volunteers to come to the front to hold up cards. The cards will have either a sound to say, or an image. Add another learner to be the conductor. Their job is to sequence the cards how they wish and to conduct them, i.e. tell them when to make their sound.

When the conductor starts to tell the learners with the cards to perform their sounds, watch what happens when they reach a learner with a picture. How did the learner translate the sound? Did they even perform it? This highlights that computers need precise instructions, whereas humans can interpret instructions without the precision needed by a computer; humans can make assumptions. Debug the sequence by removing the pictures and then play the sequence again.

If there is enough time, follow the additional instructions on the worksheet to help demonstrate to learners the concepts of selection and iteration.

RELEVANT LINKS

TCC 'Sequencing' lesson:
helloworld.cc/sequencing

Figure 1 AI in a Box

DEMYSTIFYING AI

An open educational resource collection of unplugged AI teaching activities

Everybody is talking about artificial intelligence (AI), but student knowledge about the topic is still limited. We urgently need to demystify the technology and make basic AI concepts understandable for everybody if we are to ensure informed participation in an AI-driven society in the long term. To contribute to this work and implement it successfully with different age groups and students who have no CS knowledge, or only a small amount, we present a collection of AI activities that mainly take an unplugged approach.

About the unplugged activities

These activities were designed as part of the portable AI learning lab 'AI in a Box'. The activities deal with general functional principles of AI, simulate AI applications, consider the social and ethical dimensions of AI, and give insights into AI research. The games, experiments, and activities were implemented in large wooden boxes; an example is shown in **Figure 1**.

However, as this learning lab is only available at one school at a time, and reproducing the boxes is complex and costly, we went on to develop pen-and-paper/print-and-cut versions of our activities. These are independent of the AI learning lab, and are available as open educational resources (OER) so that they can be taken to class and integrated easily into different teaching scenarios.

The activities can be used for different purposes. Some focus on introducing concepts, mostly in group work or pairs, while others serve to initiate discussion about different aspects of AI. Several selected activities are presented here, and more can be found on the website (**helloworld.cc/ai-in-a-box**).

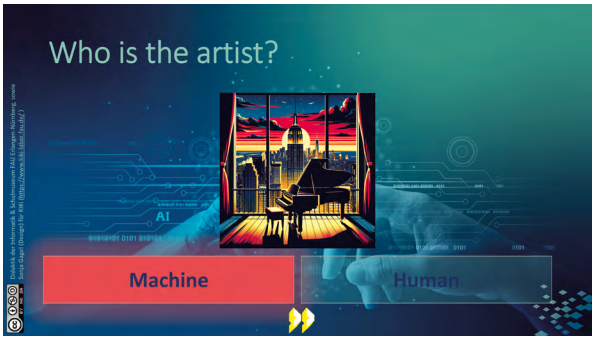
For all the activities, background texts are available that can also be used in class to guide the teaching when working with one of the activities. (HW)

ACTIVITY 1: WANTED:AI

This activity deals with the use of AI systems in everyday applications. The game introduces the topic and serves as a basis for discussion.

In pairs, students decide whether or not an object shown on a picture card uses an AI system. The cards are categorised accordingly; some of the objects shown can be clearly identified, while some cannot. The students' mappings are then discussed in class. The items that caused controversy or doubt (such as a refrigerator or a cell phone) should be examined more closely and explained.

ACTIVITY 2: ARTIST UNKNOWN



With AI-operated image generators, it is possible to create realistic images of objects, landscapes, and even people. Other creative artefacts, such as music, videos, and art, can also be generated. This activity demonstrates the difficulty of distinguishing between AI-generated and human-made artefacts. Problems such as deepfakes, and the fake personas whose images and CVs adorn social media profiles, can also be discussed. AI-generated images can be analysed in more detail, asking for example which characteristics can be used to distinguish a generated face from a real one. Afterwards, the methods used to generate such AI artefacts, for example GANs (Generative Adversarial Networks) or transformer architectures, can be explained.

This activity is carried out as a poll. The teacher presents AI-generated or real artefacts, either images, videos, or music. The students vote on whether each one is an AI-generated or human-made artefact. Voting can be done either with the help of coloured pencils or voting cards, or digitally.

ACTIVITY 3: (DIS)LIKE

AI systems have found their way into many areas of our daily lives. We encounter them at school (in learning apps), when shopping (as purchase suggestions), on streaming platforms, and in medicine.

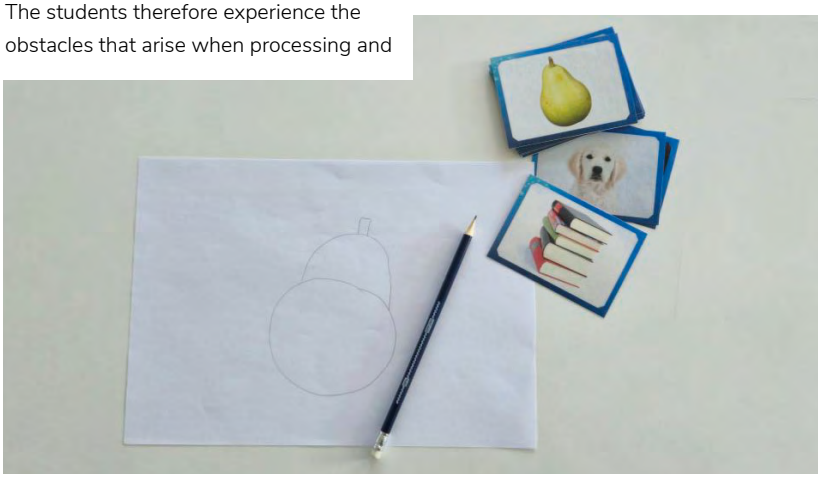
In this activity, students evaluate different scenarios for the future use of AI systems. These scenarios come from the categories 'School and work life', 'Medicine and healthcare', 'Climate and ecology', 'Freedom and security', 'Politics and law', and 'Everyday life'. The scenarios are written on reversible cards and the students decide how they feel about each use of AI systems. To illustrate their decisions, they place each reversible card either with the positive (thumbs up) or negative (thumbs down) side facing up on the game board. After the first round, they are asked to rethink their decisions, taking the view of themselves when they are 15 years older. The respective decisions and the students' reasons for them can be discussed in the class afterwards.

ACTIVITY 4: REALITY TABOO

This activity serves to illustrate the abstract ways in which an AI system can process the world.

Students work in pairs and each pair is given a set of picture cards. The game is based on the well-known game 'Taboo': one student describes as accurately as possible a picture which is kept hidden from the second student, using only geometric shapes, line types, and colours, as well as information about direction and position. The partner redraws the pictures as well as possible based on the instructions given, and finally tries to recognise the picture.

The restrictions on the description ensure that it is not possible to use complex or abstract terms, which computer systems cannot use either. The students therefore experience the obstacles that arise when processing and interpreting sensory data. For computers, such data consists of individual coloured pixels. Abstract concepts such as objects or people are not displayed as a whole, but consist of changing pixel patterns, and have no meaning to the system. In this activity, students recognise that reality must be represented in an appropriate, machine-processable manner, and that this is not always possible in any clear and meaningful way. With this knowledge, you can discuss with your class modern approaches to image recognition with the help of machine learning, and the functioning of generative networks. The representation of knowledge about the world and reality by AI systems can also be discussed following the activity.



ACTIVITY 5: ORACLE COPS

More and more law enforcement agencies are testing AI systems to predict crimes, and in some countries, these systems are already fully integrated into everyday police work. The functioning of these predictive policing systems will be discussed in this activity.

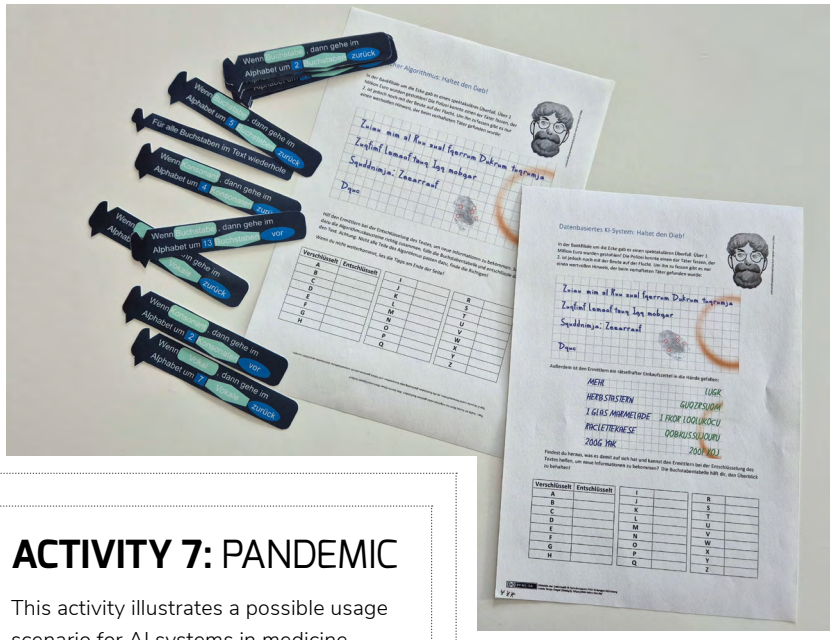
In pairs, students receive a set of incident cards, a city map, and marker tokens in two colours. The students discuss the events on the cards and decide whether they might increase the risk of a crime. Based on their decision, they lay out a colour token on the city map. Once all the cards have been processed, a possible route for the police can be planned, depending on the potential risk of a crime in the different areas of the city.

The aim of this activity is to sensitise students to the fact that seemingly objective AI systems have been trained on data that has been influenced by humans. This data does not contain a neutral evaluation, but is characterised by the subjective attitudes and views of humans. This also means that it should not be assumed that the data is necessarily free from issues such as discrimination, sexualisation, radicalism, and so on. Building on this, some of the events included in the game can be analysed further, and hidden information in socio-geographical data can be discussed, along with other critical application scenarios, reinforcement of social inequalities, and the role of bias in AI systems.

ACTIVITY 6: CRYPTO BATTLE

This activity aims to illustrate the difference between a regular algorithm and supervised machine learning (ML).

In groups of four, students try to decrypt a text which reveals the hiding place of the loot from a bank robbery. In a competition to see who can decrypt the text more quickly, two students have to compose a decryption algorithm with algorithm puzzle pieces, while the other two are given example texts of encrypted and decrypted words to deduce the encryption and decryption rules by learning from data. Three levels of difficulty are available to suit varying student ages and skill levels. After decrypting the text, students compare their different approaches and analyse the strengths and weaknesses of both sides. This should reveal how regular algorithms have to be composed by human programmers in detail, while ML algorithms can learn rules from sample data. It also shows the statistical nature of ML results, which differs from the reliable facts produced by regular algorithms.



ACTIVITY 7: PANDEMIC

This activity illustrates a possible usage scenario for AI systems in medicine.

AI systems can be used to help predict geographical risk areas for the development of a pandemic. In pairs, students take the role of an AI system in evaluating different data which, at first glance, does not seem to be linked to medicine and the development of diseases. However, it shows that socio-geographical data provides a lot of information about the overall state of a society and can also help to predict issues and risks in areas that are not directly related to the data. These aspects are discussed in detail after the activity, asking questions such as: How do conflicts or wars contribute to health in the respective countries? Why is population density a relevant factor? In this context, it is also possible to present real AI projects that have the goal to predict epidemics and pandemics and consider their actual success.

ACTIVITY 8: FESTIVAL HIDE AND SEEK

This activity illustrates the concept of Big Data with a hidden object game.

By following stories that are hidden in the picture, students trace data points in an incredibly large amount of continuous data and thus reveal information that is hidden in the appparent chaos. You can then introduce the concept of Big Data and show how AI systems use such large amounts of data to identify complex patterns. They can generate information from large amounts of data which is not visible to the human eye.

> WE NEED TO DEMYSTIFY THE TECHNOLOGY IF WE ARE TO ENSURE INFORMED PARTICIPATION

ACTIVITY 9: ONCE UPON A TIME

In this activity, the students solve bandolinos (puzzles where you match images or facts with a string) on historical facts in the field of AI. They work together in pairs or small groups and use an overview poster or smaller fact sheets to find the information needed to solve the bandolinos. Afterwards, individual events can be discussed and explored in more detail. This activity is suitable for an introduction to the historical background of AI, as well as for a discussion around the overall technical requirements of AI systems.

ANNABEL LINDNER

Annabel is a researcher in CS education at Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany. Her research focus is on AI and teacher professional development, and she enjoys developing innovative teaching materials for K-12 CS education.

MICHAELA MUELLER-UNTERWEGER

Michaela is a grammar school teacher from the greater Nuremberg area. Besides teaching CS, maths and economics, and business administration and law, she lectures at Friedrich-Alexander-Universität Erlangen-Nürnberg, and develops teaching materials and professional development initiatives for CS education.

MARC BERGES

Marc is a professor for CS education at Friedrich-Alexander-Universität Erlangen-Nürnberg. His research focus is on AI and programming education. He is co-founder of the portable learning lab 'AI in a Box'.

INSIDER'S GUIDE TO GRANT WRITING

How to unlock funding for your classroom

Ever feel like you're short on resources for your students, but unsure how to get the funding you need? You're not alone. The good news is that getting funded often comes down to one simple step: asking for it! It might seem obvious, but so many educators miss out because they don't take that crucial leap.

What it takes to get funding

Getting those much-needed resources into your students' hands is absolutely worth the effort. Here's what you'll want to keep in mind:

- **Do your homework:** this is key! You'll need to research potential funding sources to find the best fit.
- **Be an advocate for your students:** you are your students' biggest champion! Tell grant judges exactly who you are, what you need, and why it is important. They need to truly grasp who and what the funding is for and how it will enhance your students' learning journey.
- **Dedicate time to write:** the grants I've been fortunate enough to win all required a bit of time and thoughtfulness. I'd really think about what was needed and why, do my research, and even ask my students for their input (their perspective is a huge plus). Then I'd take the time to write it out carefully, have others read it over, and finally submit it.

Trust me, putting in this effort is SO worth it. You'll be thrilled you did, and most importantly, your students will reap the benefits.

Finding those golden opportunities

So, where do you even begin looking for grants? Here are some friendly suggestions:

- **Chat with your administration:** administrators receive countless emails about grant opportunities. A simple request for them to forward these to you can open up a world of possibilities.

GRANT WRITING TIPS: YOUR STEP-BY-STEP GUIDE

Ready to dive into applying? Here are some practical steps to help you succeed:

1. Do your research

- **Know your funder:** get to know the company or group offering the grant. Understand their mission and vision, and weave this information into your writing. It makes a tremendous difference!
- **Gather data on your request:** if there is statistical data supporting the benefits of what you want to purchase, include it! This shows you have done your homework.
- **Share your personal stories:** if you have used something similar in the past, personal experiences are incredibly powerful. Share that data! If you have before-and-after test data, use it. If not, start collecting it now for future applications.

2. Get your school data

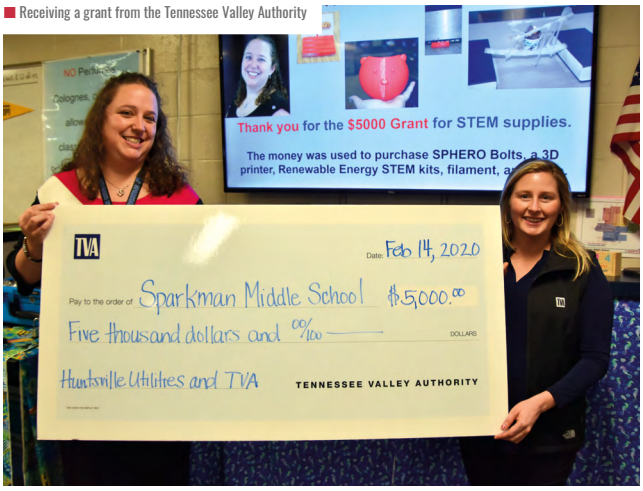
At the start of each term, I ask my registrar for school data broken down by grade level, and data for my own classes. This includes:

- Number of free/reduced lunch students
- Gender breakdown
- Ethnicity breakdown
- Total students in each grade

3. Make a budget

Even if the grant doesn't ask for it, including a budget is a great idea. It gives the grant-giver a clear picture of how you plan to spend the money. Be as detailed as you would for a purchase order (item number, quantity, description, unit price, total price). This also helps you remember what the grant was for, as you might not hear back for months. (I once waited five months to hear I won a grant!)

Receiving a grant from the Tennessee Valley Authority



- **Reach out to district personnel and local authorities:** they often have a pulse on available funding too.
- **Check your state's teaching organisation newsletter:** in the US, these newsletters are treasure troves of grant opportunities, sometimes even from the organisations themselves!
- **Join social media groups:** I'm part of many STEM and cyber groups, and they are constantly sharing grant opportunities. It's a fantastic way to stay in the loop.
- **Get involved locally:** local civic organisations frequently offer grants. Contact your local chapter of these international organisations: Civitan International (**civitan.org**), Optimist International (**optimist.org**), and Rotary (**rotary.org**). In the US, I've received funding from the Air & Space Forces Association (**afa.org**) and the Civil Air Patrol (**gocivilairpatrol.com**).
- **Don't forget Google:** seriously, hundreds of grants go unclaimed each year simply because there aren't enough applications. Try a search like 'STEM grants for teachers' and prepare to be amazed by the results! (HW)

KALA GRICE-DOBBINS

Kala, a teacher in the Madison County School District in Huntsville, Alabama, USA, was recognised as Teacher of the Year for Madison County Schools in 2024–2025 and by the Alabama State Association in 2025. She also earned the distinction of first runner-up nationally for the Air & Space Forces Association's Teacher of the Year award (**LinkedIn: Kala Grice-Dobbins, kgricedobbins@madison.k12.al.us**).

Getting those much-needed resources for your students is absolutely worth the effort of writing a grant application

4. Get fresh eyes on it (outside your school)

Have someone who isn't an educator read over your grant application. Many grants come from non-educational organisations, so it's vital that your writing is clear and understandable to anyone, regardless of their background. This also helps you to catch anything you might have missed.

5. Get fresh eyes on it (inside your school)

Next, have a colleague within your school read it. They can check your school stats and ensure the data aligns with your students. They might even offer valuable insights.

6. Keep a copy

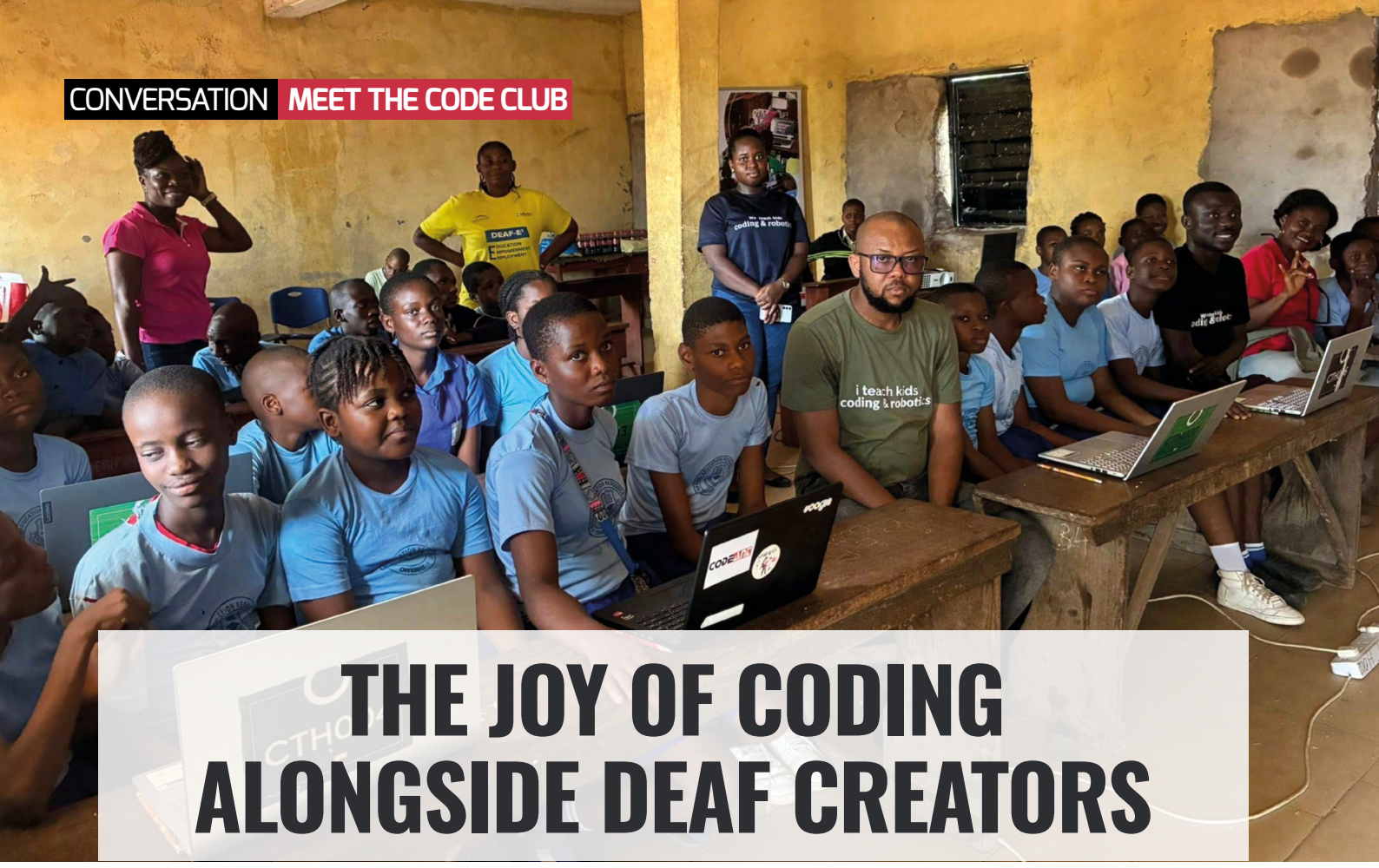
Always make a copy of your entire application and file it away. I apply for many grants throughout the year, and it's helpful to refer back to them when I finally hear the results. Plus, they can be a wonderful resource for future grant applications.

Now it's your turn!

Take some time to search and find a grant that would be a perfect fit for your class, then start drafting your application in a Google Doc (never just fill it out directly online — this way, you have a copy and can come back to it).

Over the past ten years, I've won nearly 30 grants ranging from $500 to $30,000. I'm proof that you can do this too, and the reward is being able to give your students the tools and equipment they need to further their education. For more information, you can watch the presentation I recently gave for Firia Labs (**firialabs.com**) at **helloworld.cc/grant-writing**.

You've got this!



THE JOY OF CODING ALONGSIDE DEAF CREATORS

Chidi Duru shares insights from his time leading a Code Club for Deaf children

Codeant Technology Hub (codeant.org) is a Nigerian organisation with a mission to foster educational empowerment through programming. Working across Imo State, they partner with Code Club to nurture a community of curious, lifelong creators.

Codeant took up a valuable opportunity to set up a club with a group of Deaf young people and learn alongside them. For many of the young people, the club was the first time in their lives that they had used a computer. We caught up with Chidi about his time leading the Code Club.

What were your first thoughts and feelings when you started planning your Code Club?

I was excited about the opportunity to unlock a new world for the young people — coding, creativity, and problem-solving. But I also felt the weight of the challenge: how do I teach coding to kids who are learning to use a computer for the first time, and who communicate differently?

I also knew there could be issues with the available infrastructure: old or slow computers, power outages, and limited access to the internet or learning aids. I was determined, though, because I believe that inclusivity in tech starts with giving everyone — regardless of ability — a fair chance to learn and grow.

How did you make your Code Club accessible for Deaf creators?

I approached it with simplicity, clarity, and accessibility in mind. First, I broke down the Code Club 'Introduction to Scratch' project path (**helloworld.cc/scratchintro**) into visual step-by-step guides so the creators could follow along independently.

To support communication, I had an experienced sign language teacher working with me, and I'm learning it myself.

I also integrated live demonstrations and visual storytelling. For example, instead of saying, 'Make the cat move ten steps', we used real-life gestures and visual cues, like arrows and body movement, to help understanding. We used other non-verbal cues, like raising hands to ask for help, thumbs up for 'I'm ready', and applauding (by waving raised hands) to celebrate achievements.

These simple adaptations have made the learning environment more inclusive, interactive, and fun for everyone.

What's it like to see creators finish a project?

The joy is unmistakable, even without words. Their expressions say it all: wide smiles, excited gestures, clapping, and proudly showing their screens to their peers.

Seeing them go from hesitant to confident, from passive observers to active creators, is incredibly fulfilling. You can feel their sense of ownership and pride — it's not just about finishing a task, but about building something of their own for the first time. That confidence boost is priceless.

Do your creators solve problems in unique ways that could help others?

Absolutely. Because they process information visually and often non-verbally, they naturally approach problems in creative, out-

Creators working on their projects

of-the-box ways. I've seen them troubleshoot by mimicking sprite movements with their hands, or by physically demonstrating what the code should do.

Their ability to focus on visual logic and spatial awareness is something all learners could benefit from. It's a reminder that coding is not just about syntax — it's about thinking, creativity, and expression.

Most importantly, they are learning that they can do hard things, and that mindset will serve them for life.

How has this experience changed your ideas about teaching, accessibility, and tech for young people?

I have come to deeply appreciate the power of inclusive teaching, how adjusting methods to meet learners where they are can unlock so much potential.

Technology should be a leveller, not a divider. This experience has shown me that with patience, the right tools, and empathy, every child, regardless of ability, can become a tech creator. Accessibility is not an add-on — it's a foundation for equitable learning.

What message would you share with others who are starting a Code Club for creators with diverse accessibility requirements?

Start small, but start. You don't need to have all the answers, just the heart to try and learn alongside your students. Inclusion is not about perfection — it's about presence and persistence.

These creators have so much to offer. With your support, they will show you what is possible when tech truly becomes for everyone. (HW)

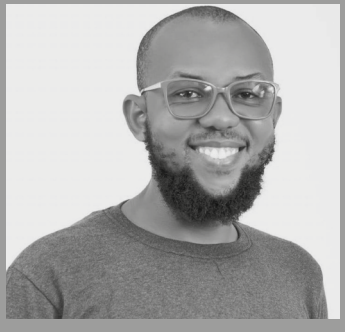
Has Chidi's story inspired you to make an impact? Join our community of passionate mentors! With active Code Clubs in over 100 countries, there's a place for you. Ready to get started? Check out codeclub.org/mentor.

Sign language teacher supporting the Code Club

> "THEIR ABILITY TO FOCUS ON VISUAL LOGIC AND SPATIAL AWARENESS IS SOMETHING WE CAN ALL LEARN FROM

CHIDI DURU

Chidi is the co-founder of Codeant Technology Hub in Nigeria, where he leads a Code Club for Deaf creators. He teaches coding, robotics, and AI to children, teens, and women, empowering communities through digital skills.



SELF-PORTRAITURE THROUGH CODE

Embracing artistry in computer science education

Even in the most highly technical, tedious work, we can't help but be creative. Deep inside our devices, scrawled in silicon, are microscopic works of art hidden by engineers for whom the pride of technical achievement wasn't enough. Silicon doodling describes a practice in which engineers hide images on the chips they design (helloworld.cc/silicon-zoo).

Perhaps this helps explain why, in my classroom, even students who love to code are most engaged by technical lessons that allow for self-expression. In one particular assignment, fifth-graders (10- to 11-year-olds) are asked to write code using a creative coding platform (p5.js; **p5js.org**) to draw shapes on a coordinate plane and eventually create a self-portrait. Calling functions, using variables, and understanding RGB (red, green, blue) colour, become an exercise in identity exploration. Some students adorn their self-portraits with contextual details, such as wearing the jersey of an athlete they admire, or making the background reflect their favourite neighbourhood park. Other students focus on the simplicity and specificity of the portrait itself, with many spending an entire class period trying to pick out the right skin tone or rearranging different shapes to create the perfect hairdo. Like most of my assignments, this one begins with an unplugged activity.

Portrait programs

Before jumping on their computers, students draw portraits of each other on paper, following sets of rules which we call 'programs'. Here's an example: 'IF left-handed, hold your pencil in your right hand. ELSE, hold your pencil in your left hand. Draw your subject for one minute, then stop. Switch hands. Draw everything you see around your subject for two minutes, then stop.' As they execute these 'programs', students practise sequential thinking without the technical intimidation of a text editor. We also use this activity as the catalyst for our discussions around identity: 'How do you see yourself? How do you see your peers? How do you want others to see you?'

TESS RAMSEY

Tess is a new media artist, technologist, and educator. They run a makerspace and teach creative and computational thinking classes to students in grades 5–8 at the Speyer Legacy School in New York, USA (**tessramsey.com**).

Students write code to create a self-portrait

Self-expression

When I devised these portrait programs, I drew inspiration from Fluxus artwork created in the 1960s. The Fluxists were known for their conceptual approach to performance art, valuing simple actions, everyday objects, and the quotidian. Their performances did not follow the formal structure of a play. Taking inspiration from musical compositions, the artists wrote poetic 'event scores', or pseudocode-style texts with predetermined rules, for their performers to follow (**helloworld.cc/fluxus-event-scores**).

At the same time that Fluxus artists were coding performances, NASA programmers working on the Apollo Guidance Computer (yes, the ones responsible for taking us to the moon) were quoting Shakespeare in the comments of their code (**helloworld.cc/nasa-code**).

By sharing stories and moments like these in the classroom, I hope to show my students that art is technical, that programmers are imaginative, and that ultimately, when we make something, we should express ourselves and have fun. Our students have a lot to say, and we should foster their artistic voice by purposefully leaning into and celebrating creativity. This is especially valid in computer science, where there is a long history of programmers personalising their work in artful and unexpected ways. (HW)

THE BEBRAS PUZZLE PAGE

Each issue, **Andrew Csizmadia** shares a computational thinking problem for your students based on the work produced by the International Bebras Community

ABOUT BEBRAS

Bebras is organised in over 90 countries and aims to get students excited about computing and computational thinking. Last November, 467,000 students participated in the UK annual challenge. Our archived questions let you create your own automarking quizzes at any time during the year. To find out more and to register your school, head to **bebras.uk**.

DOMAIN

Algorithms, logic, and programming

SKILLS

Abstraction, decomposition, and evaluation

AGE

8–14 years

DIFFICULTY RATING

Ages 8–10 hard
Ages 10–12 medium
Ages 12–14 easy

THE PROBLEM: PARKING ISSUES

In the parking lot below, cars can be parked in parking spaces or in front of these parking spaces.

If a car wants to leave its parking space, the cars that are parked in front of the parking spaces can be carefully moved forwards or backwards in order to unblock the movement of the car that wants to leave.

Example

- Car A is not blocked and can leave its parking space
- Car L is blocked by car M
- Car M must move backwards before car L can leave its parking space

Task

Select the car that needs two other cars to move forwards or backwards before it can leave its parking space.

Background

This task involves two aspects related to computer science:

1. A brute-force algorithm to search through all possible candidate cars and check which is the one that has the required property, i.e. which car can only leave its space after two other cars are pushed away.
2. The autonomous (automatic) parking algorithms, which are becoming more and more available in cars nowadays. A lot of research is done in the field of public transportation/parking systems based on autonomous vehicles. One of the advantages is that autonomous vehicle parking can be done very efficiently.

This task can be used as an introduction to autonomous systems or adapted as an unplugged activity. In addition, learners would use abstraction, decomposition, and evaluation to solve the task.

This puzzle was developed by the Bebras team in Germany, and reviewed and modified by members of the Bebras international community. The solution is on page 81.

KEYWORD SPOTLIGHT: BRUTE-FORCE ALGORITHM

Defining everyday words and phrases in computer science

A brute-force algorithm is a straightforward problem-solving technique that systematically checks every possible solution until the correct one is found, if one exists. It doesn't use clever logic or shortcuts to narrow down the search space; it simply relies on sheer computing power to exhaust all possibilities.

A simple analogy is trying to open a three-digit combination lock by methodically testing every single number sequence, starting from 0-0-0, until the lock opens.

While this method is easy to design and guarantees finding a solution if one exists, it is often incredibly inefficient and slow. The number of potential solutions can grow exponentially with the problem's size, making it impractical for complex tasks like cracking long passwords. It is most suitable for small-scale problems where the total number of possible solutions is manageable. (HW)

© Jackie Niam/stock.adobe.com

SPATIAL COMPUTING

Tracy Gardner introduces spatial computing to **Michael Conterio**

Spatial computing allows us to create 3D visualisations of real systems

Although AI is dominating the headlines, it isn't the only part of computing undergoing rapid advancements that could impact your learners. Join Tracy Gardner as she discusses one of these new technologies with Michael Conterio.

What is spatial computing?

Spatial computing is a term used to describe systems that combine the physical and digital worlds in new ways. It typically involves working in 3D space in both the physical and digital worlds.

In a video for the World Economic Forum, futurist Cathy Hackl said when describing spatial computing, "What we're talking about is the future of how humans will interact with technology. It is an evolving, 3D-centric form of computing that at its core uses AI, computer vision, extended reality, and other technologies to seamlessly blend virtual content and experiences in someone's experience of the physical world." (**helloworld.cc/spatial-computing**)

There are many other related terms such as the metaverse, extended reality (including virtual reality and augmented reality), immersive technology, and Web 4.0. These all capture different aspects of 3D technology and the future of the way humans interact with the digital world.

How is this different from physical computing?

Physical computing is only one part of spatial computing. It's the part that connects the physical world to the digital world using inputs, outputs, and sensors. For example, a spatial computing system might detect the movement of a person wearing a headset using an accelerometer. Developing physical computing skills is a key part of understanding how to build spatial computing systems.

So how does it build on physical computing to create something new and modern?

The key change we are seeing is the progress in the physical devices that we use. A few months ago I was fortunate enough to participate in a developer day with Snap where I got to try their new 'Spectacles' device, which allows you to overlay digital content onto the physical world and interact with others (**spectacles.com**). There has also been a lot of excitement around a potential new device from former Apple designer Jony Ive and OpenAI that could be the answer to 'what next after the mobile phone'. We're seeing progress in the capabilities of physical devices that we carry that could lead to new ways to interface between the physical and digital world.

Niantic created the geospatial technology behind Pokémon GO, which uses a combination of GPS signals, an accelerometer, and visual data to populate the real world with monsters that you can interact with through the app. They are now focusing on industry applications of spatial computing, including spatial planning and design, and warehouse logistics (**helloworld.cc/niantic-spatial-platform**).

Why should I be interested in this? Google made some glasses a while ago and then no one really liked them.

It's true that this is a bumpy journey. That's what typically happens when new technology is developed. Companies might not get it right the first time. One theme is hardware becoming smaller so the technology becomes more practical — who wants a huge battery pack strapped to their head?

At the same time, improvements in networks, storage, sensors, and 3D visualisation have pushed forward what is possible in industry. This includes developing 'digital twins' which allow us to create 3D visualisations of real systems, such as robots or power plants, which then make information accessible in real time.

How can we have a say in how society uses these technologies?

It's important to have conversations about how humans interact with technology and think through what kind of relationship we want to have with technology in the future. These need to include a range of different viewpoints and demographics of those who might be impacted, not just people who want to develop technology.

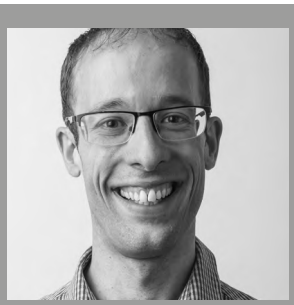
Having a well-informed citizenry will help drive governments to produce appropriate legislation and guidance to control this tech, to allow people to make their own choices about interacting with it.

What is it like to program these devices/uses? What is different from 'regular' programming?

It's really important to understand 3D concepts and how objects are positioned in 3D space. Spatial data is really important. This can be positioning objects in digital 3D space or understanding geospatial data in the physical world. This can be on a small scale such as accelerometer data detecting movement, or large-scale positioning such as GPS data. An example would be using motion data from a person to control aspects of a live theatre performance.

Does spatial computing use other insights from video games or 3D animation?

Absolutely. The industry around spatial computing builds on technology that has come from video games. At one time, 3D skills were mostly used for creating video games. Now virtual film production is common, this mixes techniques such as motion capture with digitally created 3D content. But the applications are much broader: 3D-creation technology is also used in fashion ('trying on' clothes digitally), interior design (viewing different lighting effects), architecture (walking around a proposed building), engineering (monitoring and modelling a plane in flight), scientific simulation (discussing a shared 3D view of an X-ray with a colleague), education and training (practising first aid), robotics (delivering shopping), and immersive entertainment experiences (reacting to an audience).



TRACY GARDNER & MICHAEL CONTERIO

Tracy is a computer scientist, tech industry professional, technology educator, and a co-founder of Flip Computing. Michael is a former physicist and now works as an online course production manager at the Raspberry Pi Foundation.

How can I bring this into my classroom?

Firstly it's important for young people to feel that they can influence technology. Lots of the details of how we will use spatial technology are still being worked out. It's valuable to discuss controversial topics such as whether we would want robots taking care of the elderly, or whether we want people walking around with easy-access spectacles with cameras.

Then on the technology side, you can make connections to the 3D technologies that many young people are familiar with from the games and social platforms they use. You may already include 3D-creation tools, such as Blender, in your curriculum, or run physical computing projects that collect 3D spatial data (for example using a micro:bit). When teaching using these technologies, make sure that you make links to spatial computing and how the skills they are learning are directly applicable to building the technology systems of the future. (HW)

BEBRAS PUZZLE

BEBRAS PUZZLE SOLUTION: PARKING ISSUES (PAGE 79)

Solution: Car I

Explanation

Car I is blocked by car N. There is not enough space to push car N away so that car I can leave its parking space. Therefore, car O must be pushed backwards, then there is space to push car N away for car I to leave its parking space ...

Alternatively, car M must be pushed forwards. Then, there is space to push car N away for car I to leave its parking space.

There is no other car for which two cars must be pushed away so it can leave its parking space:

Cars A, D, E, J, and Q can leave their parking spaces immediately.

Cars B, C, F, G, H, K, and L can leave their parking spaces when only one car is moved.

START AS YOU MEAN TO GO ON

Mark Crane shares his tips for a strong start to the school year

For many of us, this time of year is the start of a new school year. Our classroom management and routines are a big part of getting ready for the year ahead, but the challenges we face as computing teachers are seldom understood beyond our subject area.

Imagine the scene. Students begin to arrive outside your classroom, eager to enter and get settled in. You, like your students, are bright-eyed and bushy-tailed, having rested for six weeks or so. Your lesson is planned, the smart board is on, and you have switched on the computer and monitor at each workstation and checked they are working. You assume that each machine has access to the internet, the school network, and all the required software for this lesson. You open the door and greet students as they enter. You also have a new seating plan to roll out, a register to take, a 'do now' activity to deliver, and a reminder to give about classroom expectations. You implement the new seating plan and ask students to log in — and this is where it gets tricky.

Some students' fluency with using desktop hardware such as mice and keyboards is strong, and so are their literacy levels. For these students, logging in is a breeze. However, for many students, this is not the story we know.

Some of our students have low levels of literacy, and so entering their credentials is challenging. Most of our students have mobile devices which automatically remember their passwords (gone are the days when we could recite several telephone numbers from memory). But many students are not skilled in using a mouse or keyboard, particularly younger students who have not yet studied computing. These are not criticisms of our students; they are acknowledgements of the challenges we face today, which I am sure resonate with you too.

Our ability to manage our classes effectively in an IT suite is not just beneficial, it is absolutely necessary if high-quality teaching and learning is to take place. We need to take our classroom management seriously, and reflect on our practice, looking at the range of contributing factors that affect the start of our lessons.

MARK CRANE

Mark is the curriculum leader for computing at Sir Harry Smith Community College in Cambridgeshire, UK. He has over ten years of experience in education, specialising in the development of curriculum and professional development for experienced and trainee teachers.

Four strategies for a fruitful start

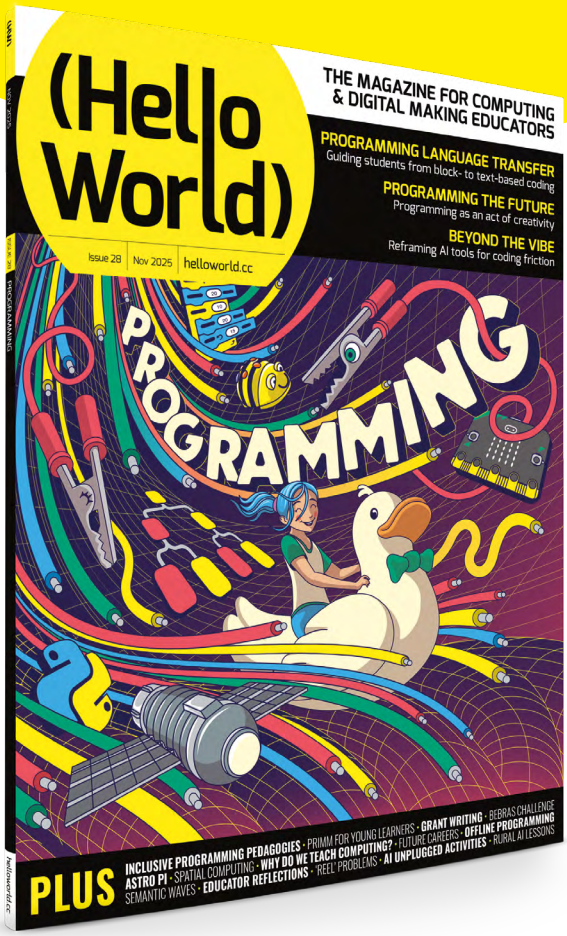
Here are my top tips for a strong start to the school year:

1. **Develop and print out a seating plan** and place students according to the plan as they enter the room.
2. **Value people**: as hard as it can be, remember students' names, and use them! This is critical for relationship building.
3. **Prepare a 'do now' activity**: display an open-ended/lateral-thinking activity on the board for students to complete while others are getting logged on (**helloworld.cc/lateral-thinking-puzzles**).
4. **Remember your IT support staff**: utilise your network manager and team of technicians. They are often prepared to be on hand to help with password resets and to assist with ensuring equipment is up to date and in working order.

I hope that you have a successful start to the new academic year, and remember to be kind to yourselves; teaching is challenging, and we are all still learning. (HW)

SUBSCRIBE TODAY

FREE IN PRINT for UK-based educators



Why subscribe?

- Never miss an issue of Hello World
- Get notified about our latest podcast episodes
- Exclusive news, the latest research findings, and in-depth features
- Free, convenient, and full of practical ideas you can use straight away

TO SUBSCRIBE VISIT:

helloworld.cc/subscribe

Prefer a print copy?

Visit **helloworld.cc/buy** — we ship to over 50 countries.

We never charge full price. Your purchase contributes to the storage, processing, and shipping of your print copy.

FREE PDF for anyone, anywhere

helloworld.cc

Raspberry Pi
Foundation