PROGRAMMING PEDAGOGY THE BEST WAY TO TEACH PROGRAMMING?

Alan O'Donohue has a collection of ideas for teaching programming to a mixed ability class of students...

"The role of the teacher is to create the conditions for invention rather than provide ready-made knowledge" - Seymour Papert.

Why should I read this guide? For those learning how to program a computer, there are many different learning routes available, but teaching programming to a class of 30 mixed ability students is a very different matter from learning how to program. This article compares a variety of strategies for teaching programming with varying degrees of success. The guide should help even the least confident among computing teachers select some strategies to support them to teach programming to their classes. The guidance within this article is aimed broadly at teachers of Key Stage 2, Key Stage 3, and Key Stage 4. Bear in mind that this guide represents the opinions and experiences of the author and is neither exhaustive nor prescriptive.

What if I'm too busy to read the full guide? In short, to develop successful problem-solvers (and programmers), teachers should plan lots of repetitive practice through a variety of investigation activities and problem-solving exercises for their classes, building in frequent opportunities for students to reflect on the successes of their solutions.

What does the word 'programming' refer to? Within the context of this article, I use the word 'programming' to refer to many more activities than just the activity of text-based programming itself. In computing education, the word 'programming' is frequently used as an umbrella term to describe a group of problem-solving activities that include computational thinking, algorithmic design, and developing a coded solution to an identified problem.

How important is it for computing teachers to be able to program? In my work supporting schools, I've noticed a tendency for some teachers who lack previous experience of programming to focus, perhaps disproportionately, on the coded element of programming. This may be because some teachers have identified this as an area where they currently lack expertise, and this induces a sense of fear or panic. To illustrate this, I was surprised when I encountered a non-specialist Key Stage 1 teacher enrolled on a 'Teach Computing with Python' course I was leading a couple of years ago. She had been informed by a senior leader that she should be teaching Python to her Year 1 class in order for them to succeed in computing and to really stretch the class!

There's a danger that placing too much emphasis on teaching students how to develop coded solutions may mean neglecting

to teach computational thinking and algorithms. Computational thinking includes a range of critical problem-solving skills that can be applied to many situations, not just those requiring a computer. In fact, many aspects of programming and algorithmic design can be taught in a manner that requires neither computers nor programming languages. For further reading, refer to CSunplugged.org (**www.csunplugged.org/en**).

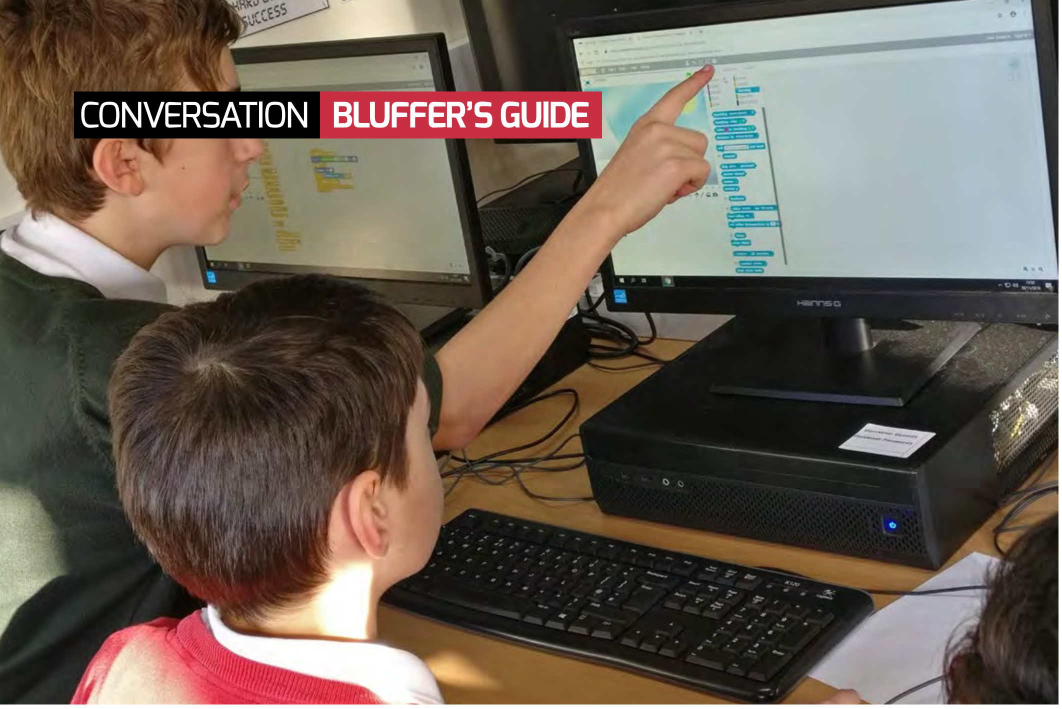
There are many stages involved in developing an effective coded solution, and many good habits and strategies that teachers should encourage their students to follow, for example identifying the essential and desirable requirements of the problem in the first place. If a student isn't clear what problem they're trying to solve, the solution they develop may not actually solve it. Equally, the ability to develop an effective testing strategy is another important skill – this helps students to evaluate how well their solution performs when presented with data other than that which is expected in order to test the robustness of the developed solution.

So, what is the best way to teach programming? There are many best ways to teach programming, since there are a broad range of different approaches and teaching strategies available, and while this guide isn't exhaustive, it highlights some of the more popular approaches that computing teachers are using. There's no single approach that will suit every teacher in every setting. There's a great deal of overlap between some of the strategies described below. Teachers are more likely to find over time that different approaches suit different settings, scenarios, and the varying needs of different groups of learners, as well as their own comfort and confidence levels. As part of their own professional development, teachers should feel encouraged to experiment with different approaches in their own teaching until they find a flexible strategy they feel comfortable with.

What difficulties are teachers likely to encounter? It's no easy feat teaching students effective problem-solving skills; it requires a large commitment of preparation, planning, effort and energy, and lots of repetitive practice with students over a long period of time. However, some teachers may find themselves in very challenging circumstances, where there are unrealistically high expectations for results, the pressures of reduced time, and an atmosphere of high accountability. In these stressful situations, there's an overwhelming temptation to simply teach students the answers to the problems or provide them with pre-prepared solutions, rather than teach the methods that students can use to get there under their own steam. While this option may appear to solve immediate issues, it will almost certainly lead to longer term, potentially serious consequences. In one recent example, schools witnessed an increase in malpractice investigations into the teaching of GCSE Computer Science when it appeared that many students had developed near-identical solutions to the same set of problems set by the exam board. While it's inevitable that students working on the same problem are bound to develop similar solutions, in some cases these students were incapable of explaining how they themselves had arrived at their solution and presented solutions they themselves didn't understand.

Additional challenges encountered teaching programming:

- **Few classes are the same as any other** – since students hail from different backgrounds and their prior experiences vary, this can have a strong effect on their levels of confidence and perceived abilities. To remedy this, build student confidence by planning to set problems broken down into smaller more manageable chunks, ensuring that the first few problems are accessible to all students.
- **Successful problem solving requires students to demonstrate persistence, resourcefulness, and initiative** – unfortunately, these qualities don't come instinctively to all students. When I've encountered this issue with some of my classes, I've put the computers away and planned whole-class problem-solving activities to highlight these qualities and to develop these characteristics within my students.
- **Lack of teacher experience** – even teachers who've been teaching computing for five years or more may only have been teaching programming for a few years, while our colleagues in other subject areas have access to a wealth of knowledge and experience of teaching their subject. Thankfully, there have been recent increases in the amount of studies being published into researching which are the most effective programming pedagogies. You can keep on top of the latest studies by reading issues of *Hello World*.



What strategies are there to teach programming? What follows below is a list of popular approaches used in different settings and scenarios; not all of them are presented as recommended strategies. Also, don't view this list simply as discrete strategies, since some of them can be blended or combined to achieve different effects. If there's one you haven't tried yet, you might decide to explore it further and share your conclusions with other teachers in your team or at a CAS Meeting:

1. **Using prepared resources and worksheets** – When students work through activities following worksheets provided by their teacher or online coding instructions on a website that teaches "how to code", often the instructions have been created by someone other than their teacher, for example "Type this in… now type this to make x happen".

- **Advantages** – There's an abundance of materials already available, reducing the amount of advance preparation required. These activities don't require a lot of support, input, or expertise from the teacher, and they don't rely heavily on teacher knowledge or experience either. When pupils encounter a situation in which their code doesn't work, it may be easier to spot the error as the resource has already demonstrated the perfect code solution. As such, if the student's doesn't work, it's because they've made a mistake copying it down. Often adopted in scenarios when teacher expertise is quite low, such as Coding Clubs led by volunteers.
- **Disadvantages** – This approach isn't particularly exciting or stimulating for students, and there's a heavier emphasis on completion of activity, rather than deep learning taking place. It's unclear how much information and understanding the children actually retain afterwards. The approach leads to students becoming heavily dependent on the provided resources and means there's often little difference between individual student work. This can also lead to problems related to assessment; when all students have effectively the same work, how can you discern which students have actually learned the most from it? In this scenario, teacher assessments may be biased toward the only one perfect solution, and it has the unintended effect of suggesting to learners that there's only ever one solution to the problem. We know very well that there are often many different solutions to the same problem, each with its own merits. Students may complain that they've only learned how to do one discrete activity, not apply what they've learned to another context.

2. **Follow the teache**r – The teacher usually starts typing in a particular piece of code and then demonstrates what it does to the class. Then the teacher explains what they're doing and asks their students to repeat the exact same exercise on their computer and test it.

- **Advantages** – Doesn't require that the teacher does a lot of future planning or preparation of resources, as the teacher can pretty much decide what the learning is shortly before the lesson. When the teacher makes mistakes, it helps the students to see that errors are an everyday part of programming.
- **Disadvantages** – This approach offers very little intrigue or surprise. The joy of discovery is robbed from the learners; if they've already seen what happens, there will be little excitement or interest value, unless of course it doesn't work when they try it themselves! Doesn't easily allow for students to progress at their own individual pace, since the students are reliant on the teacher to show them the next steps.

3. **PRIMM** – See article in *Hello World* issue 04, PRIMM - Predict Run Investigate Modify Make. This is best viewed as a set of nested activities rather than one single strategy, meaning that the teacher doesn't necessarily need to follow all five steps each time in a linear fashion or even in the same order. So, a teacher might plan for a series of investigation activities first, then ask the students to make some modifications, predict the outcomes of these modifications, and then run to see how accurate their predictions were.

- **Advantages** – The learning impact is backed up by studies as well as anecdotal evidence, which agree that this is an effective approach to teaching and learning programming.
- **Disadvantages** – None apparent, as long as the teacher doesn't slavishly prescribe all five steps be followed in strict order.

4. Step-by-step or build-up exercises – The teacher may provide their classes with a booklet of exercises, each one requiring a solution before moving on to the next, for example "First complete exercise 1, then move on to exercise 2". This approach aligns with the concept of 'Assessment 4 Learning', and helps students recognise what they can already achieve and at the same time helps to highlight apparent gaps in their knowledge, understanding, and skills.

- **Advantages** – Linear process makes it easier in some ways to track, assess, and monitor learning progress and achievement. It enables students to take more responsibility for their learning, since they can see their own progress as well as future obstacles to their own learning. Allows students to progress at their own individual pace.
- **Disadvantages** – If a suitable booklet doesn't already exist, then it requires the teacher to spend a lot of time creating the resource.

5. Challenge based – The teacher sets a number of stepped challenges for the class, which increase in difficulty and challenge level. All the class may be working on the same theme or context, but working on different levels of complexity of the problem at the same time. The challenges may outline broad problem scenarios, with opportunity for students to solve parts of the scenario in stages.

- **Advantages** – A range of possible outcomes enables students of all levels to achieve an element of success. The different solutions that students ultimately arrive at provide a valuable resource for critical appraisal.
- **Disadvantages** – Requires a lot of teacher preparation to ensure that the challenges match and stretch the abilities of the class.

6. Modifying existing programs – Similar to the PRIMM approach described above, the teacher starts by directing the class to examples of programs that already exist, often these may be games. Then the teacher asks the class to predict the effects or what may happen if certain parts of the program are modified. The class then test their theories. The teacher may then ask the class to try to solve other challenges by modifying other parts of the existing programs to achieve varying outputs.

- **Advantages** – When using existing programs, there isn't a lot of resource creation required. The students can find this approach very stimulating, especially since they haven't had to start with a blank screen and create their own program. When errors creep in, the students can refer (or revert) back to the original provided program. Students are generally quite engaged in this approach as there's a fun element in changing and modifying existing code, particularly as it doesn't rely on the student understanding all of it.
- **Disadvantages** – It requires a lot more thinking and planning ahead if this approach is being used regularly to support a list of intended learning outcomes in a heavily structured manner.

7. Paired programming – See the article in *Hello World*, issue 04. May be used in combination with some of the approaches listed above. Students are placed in matched pairs to arrive at a solution together. Each student is assigned the role of driver or navigator and they take on each role in turns at timed intervals while working towards common goals.

- **Advantages** – Over a period of time, it takes a lot of the pressure off the teachers, since students naturally support each other, and build up their confidence and problem-solving abilities in pairs. Students are less prone to make errors as there's more than one pair of eyes reading the code on screen. A lot of problem solving takes place 'off-screen' in the discussions between the pairs.
- **Disadvantages** – At the beginning, it requires a lot of explanation and enforcement until the students become accustomed to working in pairs and other good habits. Intended as a way to support student learning and progress, some teachers haven't easily identified ways to assess the student learning progress, but this is resolved if the paired activity is viewed as learning rather than an assessment opportunity. (HW)

FURTHER READING

PRIMM - *Hello World*, issue 04
Pair Programming - *Hello World*, issue 04