Laying Firm Foundations

A conceptual approach to programming

Structured Programs

Session Notes

Background to Laying Firm Foundations

Most pupils arriving in secondary schools will be familiar with the notion of algorithms as a sequence of steps. They will also have had some exposure to writing simple programs. They will probably have encountered sequencing instructions, and used repetition and conditional expressions. At Key Stage 3 we need to emphasise that algorithms work hand in hand with data structures. Pupils will have less exposure to this idea. To provide both continuity and progression at KS3, practical programming needs to provide repeated exposure to the same constructs and basic structures (variables and arrays) in a variety of contexts.

Moreover, teachers need to have sound reasons to engage in such activities. It is not enough to teach programming because the National Curriculum decrees it. We therefore want to consider the bigger picture; the broader educational value of learning to think like a computer scientist, of which programming provides one practical expression.

The problems posed and their practical outcomes should reinforce the key constructs they are familiar with from primary schools but as programs become more complex, so there is a need to emphasise decomposition as a technique by which complex problems can be broken down into their constituent parts. Structured programming can then introduce procedural abstraction whereby code sequences can be defined as named functions and procedures. Once defined, the detail can be hidden; a function or procedure will perform its task when called by name. Structured programming – developing the idea of top-down design, decomposition and abstraction – is a key element in equipping children to tackle more complex challenges.

Once familiar with these foundations, the extra demands presented by text based programming can be addressed. There is a move to introduce text based programming very early (even in primary schools) but visual languages provide a much lower floor of entry, and can be used to good effect right through secondary school. Presenting the same constructs in a visual and text based language can help make conceptual connections.

Learning to program is difficult. There is a heavy cognitive load. This Unit is the first of two with a strong emphasis on concepts that run, like a golden tread through approaches to all programming challenges. It urges a staged approach, with each activity having limited goals to minimise potential confusion. It uses ‘toy environments’ to focus on specific constructs or concepts, whilst providing motivating contexts.

For many ICT teachers, teaching programming will be a new challenge. Even those experienced at teaching Computing to older students will face new challenges making the ideas accessible to younger pupils. Over the next few years new subject pedagogy will emerge from this collective experience. A key aim of these sessions is to encourage discussion around what we are doing and what works best.

Laying Firm Foundations: A Conceptual Approach to Programming is the full Tenderfoot Unit. It comprises three separate sessions;

- Concepts and Continuity
- Structured Programs
- Transitioning to Text

Structured Programs

A central idea that goes hand-in-hand with algorithms is the notion of data structures. Algorithms + Data Structures = Programs is an old, but well known book written by Computer Scientist Nicklaus Wirth. The title summarises the challenge facing programmers. They need to develop algorithms which manipulate data. The preceding Session had a focus on the three key constructs for building algorithms (sequence, selection and repetition). This Session considers how data is held in a program, that is, the Data Structures in Nicklaus Wirth's equation. It emphasises a simple, but absolutely key concept: the notion of a variable. This is the idea that any data we wish to hold in a program must be put into a named 'container'. This is the simplest data structure, on which many more can be built. It is therefore a key idea for students to grasp. Children don't intuitively 'get this' – it is an abstract concept, and how we introduce it can have a huge bearing on children's understand.

Using a visual language, a quick challenge to draw a 'squiral' provides the foundation for the second, main activity Wacky Races. This draws together the key concepts and constructs that are central to developing any computer program. Once again, pointers and resources are also provided for further activities, four in total, which seek to reinforce correct use of variables and highlight the importance of decomposition and abstraction as a vehicle for building more complex programs.

The aim of this session

It is important to be explicit about the intended outcomes for attendees. Do not assume they will be aware of the purpose of your session. Consider too, any prior knowledge they may need for the session to succeed. By stating the intentions, it helps avoid getting sidetracked into discussing the detail of individual classroom activities. The primary aim is to ***educate teachers*** and illustrate the breadth and depth of Computer Science. The specific outcomes ***for teachers*** from this session, are to

- Recognise the three key constructs used in every algorithm
- Appreciate the centrality of a variable
- Become more familiar with functions and procedures
- Recognise their role in structured programming.

The purpose of Tenderfoot is to equip ***trainers*** with resources to broaden the outlook of teachers new to Computing. The intention is to provide a buffet of resources on which teachers can draw, to enrich their Computing lessons, at the same time as meeting the key aim: providing greater depth of knowledge for ***teachers*** themselves. Developing ***teachers*** is the focus, *<u>not</u>* providing activities for pupils or suggested schemes of work. It is up to teachers themselves to judge what might be appropriate for their particular classrooms, and at what age activities might work best.

The session aims to stimulate debate amongst attendees about both subject content and associated pedagogy. It ends with an encouragement to

- reflect on classroom practice
- consider the potential for engaging in more formal action research and
- achieve accreditation through the BCS Certificate of Computer Science Teaching

Throughout the material there are references to famous computer scientists. The aim is to encourage teachers to delve deeper and take the ideas further. Before delivering the session, please check you are comfortable with the narrative and rehearse delivery to familiarise yourself with transitions.

A practical programming challenge and two supplementary projects which focus on developing the ability to decompose problems and creating solutions using procedural abstraction. The second project is provided as an exemplar investigation, scaffolded to encourage children to read code.

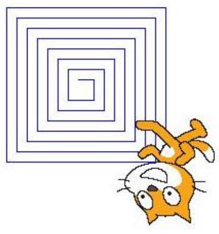
Preparation required:
Container and mini whiteboard to demonstrate variables.
BYOB installed (goo.gl/biqgp3), media files for Hungry Frog activity

Variables and Data Structures

'Algorithms + Data Structures = Programs' is a well-known book written by Computer Scientist Nicklaus Wirth. The book title summarises the challenge for programmers; developing algorithms which manipulate data. The presentation considers how data is held in a program, introducing an absolutely key concept: the notion of a variable. This is the simplest structure, on which many others can be built. Don't assume children will intuitively 'get this' – it is an abstract concept, and how we introduce it can have a huge bearing on understanding. The presentation includes a clip of Alan Kay from 2007. The whole talk, about the power of programming to inspire thinking and foster curiosity, is at goo.gl/EWsbsM. Children discover best the properties of variables through practical challenges. They learn by doing, by observing and building their own mental model of how variables work.

The presentation allows students to predict what the Cat will say. Some may expect it to say 'counter' if they have never come across the notion of a variable. Others may intuitively see what will happen but not have the language to explain. Ask students to read the code on the screen line by line. If the answers are imprecise, model the language required e.g. 'A value of 1 is assigned to the variable called counter', 'The contents of the variable called counter (which is currently 1) are displayed in the speech bubble for 1 second.' etc. Conceptually, a variable is a container, with a label, into which a value can be placed. The value can change, but the container will only ever contain one entity. Acting this out with a box and a mini whiteboard slate, on which pupils can rub out the old and write new values, before putting it back in the box can help. If the children think they have 'got it' ask them to chant the steps, before enacting the animation. It is surprising how often they get it wrong!

Drawing A Squiral



Some children will grasp the idea of a variable very quickly, but for many, repeated exposure to the concept is needed. Drawing a squiral provides a good lesson – with plenty of extension challenges, along with some powerful visuals. Discuss how it might be drawn (from the centre outwards), using a loop and variable. For the more able, ask pupils to investigate changing the amount incremented on each step. They can then control the gap between the lines. Returning to the centre, within the squiral involves recognising how far to move, before redrawing a decrementing squiral. A final challenge could involve creating a 'gap' variable, and refactoring the move at the edge, to reflect half the value of the gap. This is a good example of how code can be refactored, spotting the pattern between the gap and the move, to make it more general.

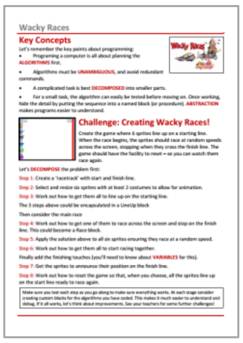
Many of the tools and techniques introduced won't be new to pupils, but in secondary school we need to put what they know into a conceptual framework, constantly reinforcing that these key concepts lie at the heart of developing Computational Thinking. Programs are built up of algorithms acting on data structures. The key to handling complexity lies in decomposing big problems into smaller ones, and that once a small program is created, the detail can be abstracted away by developing functions or procedures. So complex programs are built by assembling smaller blocks of code. A challenging activity illustrates these key concepts and builds on the ideas of decomposition and abstraction.

Wacky Races

An introductory slide shows a finished program running. As it runs, point out the start and finish line, and how the six sprites line up to start when the Reset button is pressed. The race is random and the outcome will differ each time. Discuss how we might break down the task. Try to draw out 3 or 4 distinct areas to develop independently e.g. Race Setup, Actual Race and the Finish. Each can then be further decomposed:

- When considering the Race Setup it's worth noting the co-ordinates used on the screen, so the six sprites can be spaced equidistant.
- Again we can decompose the Main Race. We can consider using a loop to repeat a move command, changing the costume each time to create the animation. A selection statement (IF touching) would stop the movement when it touches the red line.
- A counter variable could help keep track of position, incrementing by 1 each time a sprite finishes.

Once we have decomposed the problem, we can think about naming procedures to encapsulate the code. The race breaks down neatly into the commands to Line Up, which can be called by clicking on the Reset Button, and the commands to Race. Procedures to accomplish particular tasks can extend the functionality. Three possible extensions are suggested. The last one is challenging but doesn't require any more knowledge of structures than that covered.



Wacky Races

Key Concepts

Let's remember the key points about programming:

- Programing a computer is all about planning the ALGORITHMS first.
- Algorithms must be UNAMBIGUOUS, and avoid redundant commands.
- A complicated task is best DECOMPOSED into smaller parts.
- For a small task, the algorithm can easily be tested before moving on. Once working, hide the detail by putting the sequence into a named block (or procedure). ABSTRACTION makes programs easier to understand.

Challenge: Creating Wacky Races!

Create the game where 6 sprites line up on a starting line. When the race begins, the sprites should race at random speeds across the screen, stopping when they cross the finish line. The game should have the facility to reset – so you can watch them race again.

Let's DECOMPOSE the problem first:

Step 1: Create a 'racetrack' with start and finish line.

Step 2: Select and resize six sprites with at least 2 costumes to allow for animation.

Step 3: Work out how to get them all to line up on the starting line.

The 3 steps above could be encapsulated in a LineUp block

Then consider the main race

Step 4: Work out how to get one of them to race across the screen and stop on the finish line. This could become a Race block.

Step 5: Apply the solution above to all six sprites ensuring they race at a random speed.

Step 6: Work out how to get them all to start racing together.

Finally add the finishing touches (you'll need to know about VARIABLES for this).

Step 7: Get the sprites to announce their position on the finish line.

Step 8: Work out how to reset the game so that, when you choose, all the sprites line up on the start line ready to race again.

Make sure you test each step as you go along to make sure everything works. At each stage consider creating custom blocks for the algorithms you have coded. This makes it much easier to understand and debug. If it all works, let's think about improvements. See your teachers for some further challenges!

The handout is useful for reference. However, it is often difficult for children to code from a blank canvas. If creating Wacky Races from scratch would be a 'step too far' you could provide a partial implementation and set challenges to modify the existing code. Several incomplete versions of the program are included in the resources.

- Race01: provides the background start / finish lines and has sprites placed so pupils can investigate their starting position, prior to writing the code to line them up.
- Race02: models the code to return a sprite to the start, and displays the co-ordinates to make it clearer.
- Race03: develops the code for one sprite to race.
- Race04: models the encapsulation of the race sequence into a block.
- Race05: displays the use of a position variable to allow a sprite to announce its place at the end of a race.
- Race06: introduces buttons and broadcasts.

Other Projects

An excellent resource with comprehensive notes for tutors, supported by screencasts is 'Itching For More'. Available from goo.gl/xpM1Gc , the material, produced for The Royal Society in Edinburgh is part of a larger collection. Included in the resources are some ready to go materials based around one of the projects: The Hungry Frog. This requires some media files (included) which need to be copied, but is well pitched for KS3, developing the ideas and concepts we've outlined. The activity focuses on students investigating a working game and answering a series of structured questions (Task A). It then involves students analysing a part working game and comparing it to the previous example to identify the errors. The focus is on observing behaviour, then reading the code. Task C plans how to fix it. It includes a teacher presentation, two activity sheets and a homework.

Finally, a presentation includes instructions and scripts demonstrating how to play the classic game, Pong across two machines. One big challenge for Computing teachers is providing challenges for able students that don't require extra support. This is an excellent project to get able students thinking, and having a sense of doing something special. It doesn't involve any extra knowledge, apart from how to enable the sharing of sprites across machines.