Laying Firm Foundations

A conceptual approach to programming

Transitioning to Text

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

Transitioning to Text

This final Session in Laying Firm Foundations analyses the cognitive load involved in stepping up to writing text-based programs. It suggests a staged approach, starting with writing html; a set of simple mark-up instructions. The practical challenges here use a tightly restricted environment, RoboMind, which removes the need for variables and focusses solely on the algorithmic constructs of sequence, selection and repetition.

The initial activities stress the need to read code and develop the language to articulate it precisely. A further emphasis is put on ensuring code is laid out correctly. Flowcharts are introduced as an aid to visualising simple constructs. Using flowcharts the two different loop constructs, counter-controlled and condition-controlled, are highlighted.

There are four core activities. As the challenges get harder, nested loops are introduced, procedures defined and links made with earlier block-based work. Only when this is secure are variables re-introduced, this time as parameters to generalise procedures, so introducing the idea of different variable scope.

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

- Apply the key concepts required to write programs
- Recognise the three key constructs in every program
- Become familiar with flowcharts articulating the constructs
- Have considered some challenges posed by text based coding.

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, *not* 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

As with the previous sessions there are references to the work of Seymour Papert. The slides contain some animation that require careful explanation. Detailed notes are included with the slides. Before delivering the session, please check you are comfortable with the narrative and rehearse delivery to familiarise yourself with transitions.

Transitioning To Text

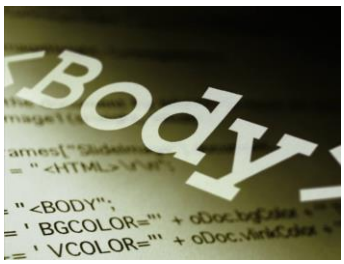
Activity Notes to support Transitioning to Text. Session 3 from Laying Firm Foundations: A conceptual approach to programming

A series of structured exercises using RoboMind freeware – a tightly defined toy environment that helps bridge the transition to text based languages, whilst reinforcing key concepts.

Preparation required:
RoboMind freeware installed (windows only): goo.gl/oNVpX1
RoboMind map sheet and wooden block for each student.

A Staged Approach

Once students have some grounding in computational concepts we can consider some of the barriers involved in transitioning to text based programming. A staged approach can help avoid some pitfalls … or at least help children over the biggest hurdle of all, commonly known as the 'syntax barrier'. First and foremost, the biggest barrier for most children is their inability to write / type accurately. An emphasis, right from the start on accuracy and attention to detail is essential.



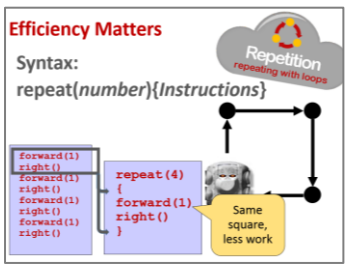
Many pupils will have some experience of html. Exercises creating simple web pages in Notepad or Notepad++ can help instil the importance of accuracy. Pupils get immediate feedback on their code. It either displays as expected, or it doesn't. Moreover, being a mark-up language, there is no extra cognitive load involving our 'big 3' constructs. A simple html document is a set of display commands, executed sequentially. With such ease of creation, they are an ideal introduction to the rigour of text based coding.

Introducing RoboMind

Given the extra demands raised by text based programs, finding motivational contexts is important. Robotics provides one example. The exercises that follow use a simple simulator and can provide good grounding for developing more complex work with real hardware later. RoboMind is a commercial product. However, earlier versions, with more limited functionality are available on a GPL license (goo.gl/oNVpX1), free to download and use. Developed at Amsterdam University, the simple simulator provides an excellent way to transition to text based programming, and paves the way to further robotics exploration using industry standard languages.

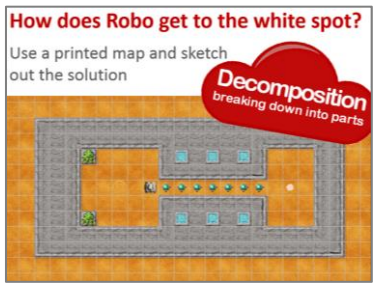
Robots in the real world provide a good introduction. iRobot produce a range of robotic devices. Their advertising campaign (2013) for the Roomba vacuum is a fun introductory video (youtu.be/mThf1LjWo1c) to engage interest. The Amazon Robotics website (goo.gl/S57UxL) features a video about people working there. Useful for demonstrating the type of careers Computing may lead to.

Robo can move forward, back, left and right. It has the facility to 'see', pick things up and paint lines. It moves from tile to tile in its world, defined by a map. The Remote Control option is a useful way to see how Robo moves and how the clicks are translated into code. It generates a command for each single step, which can then be tidied. This simple introduction ensures children can save and open files before writing a program.



When writing instructions, layout is important. We also consider code efficiency, using the usual example of a square to introduce repetition. Syntax may be a new term, so an early focus brings it to the fore. Using a repeat command as an example, highlight the command and parameter, then the sequence of instructions within braces that repeat. Notice how we put braces on separate lines. Insisting on this layout is a key point to emphasise. It makes the code easier to read and debug.

Pass The Beacons



A first exercise in decomposition uses a map and small wooden block to simulate Robo, (draw on front/back/left/right as reminders). Work out an algorithm to Pass the Beacons. Start by moving one object out of the way and jot the commands down. Do similarly for the next object, in a new column. As they work through the moves awareness of a repeating pattern may emerge.

There are many 'worlds' with the default installation. Loading a different map file change Robo's world. With the correct map and a solution sketched out try the practical challenge. Be explicit about the purpose of the exercise; for students to recognise and lay out a loop correctly. Finding repeating patterns is an example of generalisation or pattern recognition. The extension is challenging but by focusing on spotting repeating patterns they may manage it! A solution is included.

Establishing The Basics

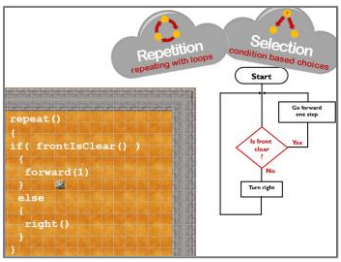
There are only three constructs for building programs. In the surface that seems simple but the real power comes once constructs are combined or nested within each other. Once constructs combine many children struggle to understand, and even those who do, find it difficult to articulate. Don't try to run before pupils can walk. Good programming requires a very solid grasp of basic ideas; endless exposure to the simple, 'one level' constructs. Don't move on to more complex challenges unless you are confident this is the case.

Even with working code, asking pupils to explain it will reveal the true thought processes involved. Reading code snippets aloud, is a good precursor to writing code. Explaining why code doesn't work reveals the ability to articulate logical processes. Asking children to read code examples and predict what will happen gives teachers an immediate measure of understanding.

Much scaffolding is required to introduce compound constructs. Establishing careful ways to lay code out helps enormously. We can model this by considering how to get Robo to walk, introducing flowcharts as a way to visualise program flow.

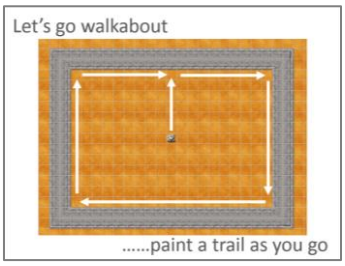
As well as moving and picking up/putting down, Robo can paint lines and 'see'. Robo can see if the front, right or left is clear, white, black or contains an obstacle. Pupils can select any command from the Insert menu, avoiding many syntax errors. Once inserted, editing and indenting helps focus on structure, rather than syntax. We consider how Robo might walk and avoid obstacles by combining a selection construct with a 'see' command. This requires careful explanation. The presentation builds the code in stages. Insisting on this sort of detail in layout is the key to clear comprehension.

When using code samples, ask pupils to predict what will happen. Building a mental model of what to expect is essential for understanding. Reading comes before writing. When the code is run, real learning starts to happen if the observed behaviour contradicts the prediction. At that point, the student has to reach for their mental toolkit and reason about what they are observing. Take time to explain the rationale – children are learning how to learn.



The code shown moves Robo 1 step if the way ahead is clear, otherwise turns right. Combining constructs is challenging for children, but building complex structures in this way makes it easier to comprehend. We now need to repeat the whole 'one step' process by putting it inside a loop construct. If students struggle, a flow chart alongside can help with reading the code, line by line. An animation is provided; emphasise that the step forward repeats many times until the way forward is no longer clear.

Going Walkabout



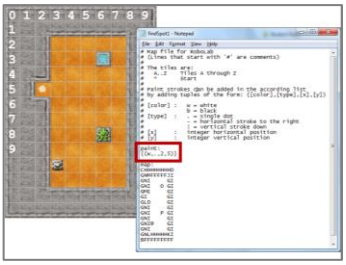
Decomposition applies at many levels. Insist on decomposing selection and iteration structures into separate parts, then further editing to lay brackets out on separate lines. Finally, use indentation to make the layout as clear as possible. This is particularly important when developing constructs that are nested within each other.

Armed with that knowledge the practical challenge is presented to code Robo to go walkabout and paint a trail as it goes. Once implemented, another problem should become obvious … Robo will continue forever. Encourage students to study the flowchart; there is no obvious end, or termination.

Extra challenges are posed to stop Robo in the top right corner or on completion of a circuit. These test understanding of flowcharts. If students struggle, hints and solutions are in the presentation. A good test might be to explain the impact of moving the conditional. Often answers reveal how little understanding children gain 'first time round'. Even with a grasp of the logic behind the program flow, the challenge still remains to translate that logic into code. The presentation allows the code to be built as a class question and answer exercise. Slide notes suggest the narrative.

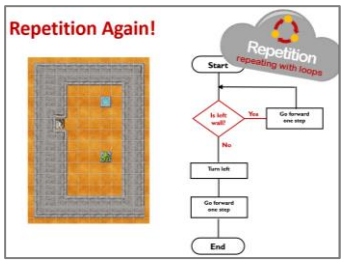
Under The Hood

A repeating theme is the time it takes for children apply basic constructs in programs but some 'get it' quickly. Having 'low maintenance' challenges on hand for these students is an essential part of differentiation. Investigating Robo's maps is itself a good exercise in computational thinking. Load a map file and it displays a new world. But how does it do it?



We can get 'under the hood' by opening it in Notepad, showing a description in another language. Let able students study it without prompts. Systematic investigation is needed to identify all the elements involved. In particular, the syntax involving for adding paint strokes requires understanding how the x and y co-ordinates are specified (numbered from zero from top left). The map shown is included in RoboMind: FindSpot1. As a challenge, students could study the maps and create two new variations as shown.

Finding The Spot

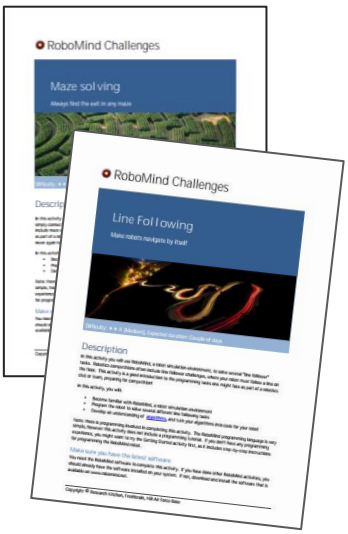


Challenge students to flowchart the sequence to find and move onto the spot. We want something similar to the walkabout code, using selection and iteration. That way we can generalise the code. Robo should find the spot, regardless of which map (findSpot1 or findSpot2) is used. The presentation builds up the flowchart, allowing the teacher to talk through the sequence of events.

Loops are also considered in more detail. Up to now, loops have been counter controlled, that is 'For' loops, or 'Forever' loops (in the absence of a number). This time we use a 'While' loop; repeating whilst a condition remains true. Emphasise the two types with a little chant for the class. 'There are 2 kinds of loop – what are they?' 'Counter controlled and condition controlled' comes back the reply … but it can still be a source of confusion. Students may have used a 'Repeat – Until', or 'Forever – If' in Wacky Races, and it is worth highlighting these. Neither uses 'while' but both use a condition to control the program flow.

Students can now program a solution and once they succeed, consider further generalising so it works on findSpot3. A further test could use findSpot4 (requiring no modification) and findSpot5 which considers a different starting position. This is quite tricky and should provoke a lot of thought and discussion. The obvious solution would take Robo forward to a wall, at which point it turns right, invoking the wall following routine. Looking at findSpot6, should illuminate an obvious problem with wall following.

Big Challenges

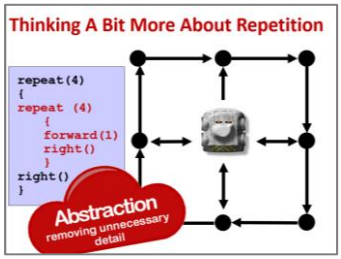


Two advanced challenges are provided, firstly a maze. If the maze is simply connected (i.e. has no loops or inaccessible areas) a wall following algorithm can provide the basis for solving the maze. This is a 'classic' challenge and can leave a lasting impression. Unfortunately, too often, it is introduced without the necessary scaffolding. With the grounding we just introduced, it shouldn't present too much of a step change. The good folk at RoboMind have produced a nice handout about Maze Solving which could encourage further research into maze solving algorithms (and types of mazes). For more able students, implementing Tremaux's Algorithm would be a good challenge.

Amazon acquired Kiva Systems in 2015. They produce robots used in automated warehouses. The Business Week video (3.30 mins) is well worth watching (youtu.be/6KRjuuEVEZs) to give a sense of how robots are used in the 'real world'. They also look rather like the Robo! Ask pupils how the robots find their way around the warehouse. The complex software is obviously important, but the physical steering is accomplished by 'line following'. You can see the tracks on the warehouse floor. RoboMind have produced a very good handout about Line Following. This includes a series of increasingly difficult challenges, from a simple line follower, to considering avoiding debris and walking grids. To tackle the later challenges you need to introduce procedures, like those in BYOB.

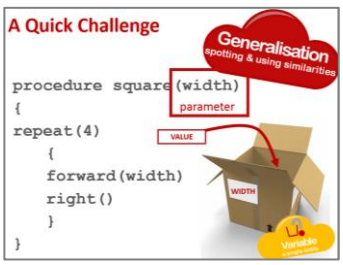
Introducing Procedures

A brief introduction, using our earlier square example follows. In drawing simple shapes we are taking directly from the ideas of Seymour Papert and exploratory Logo activities advocated in Mindstorms. Drawing patterns and shapes is a rich environment for early programming. The visual output means immediately seeing if a program works. It is not just the medium that is important, but the pedagogy too. You can get a flavour of this from the wonderful archive footage in the presentation.



Asking how Robo might create a window with 4 quadrants should provoke discussion. If not encourage decomposition, subdividing the problem. As Papert notes though, students learn best through doing. Encourage students to try, before discussing the solution. This is a repeat of a square. Using the example point out the potential for confusion unless 'nested loops' are indented. To reiterate - laying code out well is important for debugging.

It is best to avoid the complexity of nested loops in early programming. However encapsulating nested code in procedures makes it much easier to comprehend, since we can abstract away the unnecessary detail – in this case, by creating a procedure to draw a square, to remove the detail of the inner loop. We can make a link with previous work in BYOB, which helps understanding. In BYOB, we encapsulated code to draw a square in a named block. We want to do the same thing in Robo's code. As with BYOB, there are two stages, procedure definition, and (once defined) procedure calls. To define code we use Insert to avoid syntax errors, but as before, we want to lay the code out on separate lines to make the structure more obvious (as shown). To call the procedure we simply insert the procedure name in our code, just like in BYOB.



The procedure defined draws a fixed size square, but we can generalize so it can be called with different values. All squares use the same commands. What differs is the length of the side. The more general procedure definition has a parameter included in the procedure header, referenced in the body. We can now call the procedure with different values as the parameter. We can think of parameters as variables local to a procedure, like a box, named in the definition, into which we can put different values, when we call the procedure.