Laying Firm Foundations

A conceptual approach to programming

Concepts and Continuity

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

Concepts and Continuity

First we consider the bigger picture – why teach programming at all? Secondly we need to consider the fact that students will come from primary schools with very different experiences and exposure to Computing. Seeking to address the challenge of that Key Stage 2 to 3 transition, this Session starts with a brief history lesson looking at the development of visual languages such as Scratch. It traces their origins back to the pioneering work of Seymour Papert and Logo, emphasising his approach of learning through doing.

The starting point for most teachers new to Computing will, inevitably be the National Curriculum. It's therefore important that teachers at KS3 are aware of expectations in the primary curriculum, which are only likely to be partially met in the near future. Implementing algorithms as simple programs is an expectation right from KS1. At KS2 some of the key concepts underpinning programming are made explicit. These lie at the heart of the mental toolkit we wish to develop in children throughout their time in school. So this Session locates all programming challenges within the four conceptual pillars of Computational Thinking: algorithms, decomposition, abstraction and generalisation. Both activities emphasises three key constructs in imperative programming: sequence, selection and iteration, whilst also introducing functional abstraction, creating new command blocks to make complex programs easier to build. The main activity, Making Music, emphasises this by presenting a programming challenge using a block based language to write the code to play a variety of rounds. Supplementary activities from the Hour of Code, using Lightbot, are also included.

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

- Understand the key concepts required to write programs
- Recognise the three key constructs in every program
- Become 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, *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

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.

Concepts And Continuity

Activity Notes to support Concepts and Continuity. Session 1 from Laying Firm Foundations: A conceptual approach to programming

A practical introduction to BYOB (Snap!), introducing key concepts and constructs. Pupils discover the problems inherent in unstructured programs and develop techniques for handling complexity.

Preparation required:
Headphones, BYOB installed (goo.gl/biqgp3), Hour of Code resources available in browser.
Making Music activity sheets 1 & 2 per child. Hour of Code solutions available for teacher.

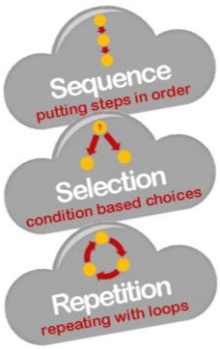
Visual Languages

Scratch is an educational game changer. Most primary children will have had some exposure; a visual language with a very low floor of entry, it removes the need to type commands. It has spawned many variants. One such variant, used initially to teach undergraduates was BYOB (Build Your Own Blocks). An online version (Snap!) is being developed but currently cannot handle music, so we use the installable version. Ensure pupils can open BYOB and save a file correctly. Use the introductory slides to stress the similarity with Scratch. Point out the 'hat' blocks which act as the trigger to execute the commands. The introduction allows computer keys to act as triggers to play notes.

Key Concepts, Core Constructs

The focus is on drawing out concepts and stressing continuity. Aim to constantly reinforce 1 big idea – which is about developing Computational Thinking. The two activities, will introduce 3 key constructs to illustrate 4 central concepts. Before starting, one final point on pedagogy; like learning to walk, children learn through doing. Nobody can 'tell' a child how to walk – they learn by falling over … again and again.

So it is with programming, but a challenge facing teachers is managing that within a class. A parent may pick a child up, but teachers can't pick 30 children up all the time. Paired programming can be an effective approach. It allows discussion and develops the ability to articulate thoughts. One child constructs the program from the instructions of another.



Programming need not be intimidating- on fact it is based on just three simple ideas, or constructs. We could call them the 'big 3':

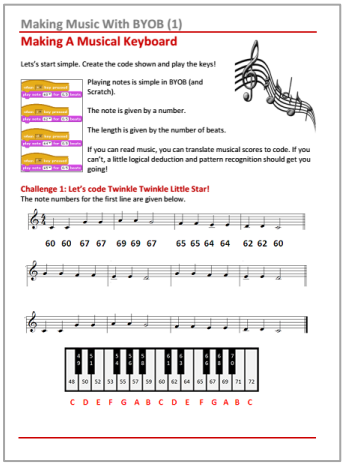
- Sequence – the challenge of arranging instructions into the correct order.
- Selection – allowing a program to branch down a different route IF a certain condition is met.
- Repetition – allows sequences of commands to be repeated by looping back.

It's useful to repeat these constructs regularly and often. Perhaps get children chanting them in response to a prompt; "There are only 3 constructs in programming. They are?" The list also represents the order of complexity of the constructs. Children generally find conditional IF statements easier than loops. Throughout any coding project, take the opportunity to highlight code and ask what construct(s) are exemplified.

We also wish to focus on key concepts, so get them upfront right from the start. Obviously there will be a focus on algorithms as a sequence of instructions, but we also want to highlight the idea of decomposition, breaking a complex problem into smaller parts as a way of managing complexity. Generalisation may be a new term for many. In a KS3 context this often amounts to trying to spot patterns. Finally, at the heart of Computational Thinking is the notion of Abstraction, which is worth highlighting as it arises.

In any early KS3 projects take every opportunity to highlight examples of both the key constructs and the core concepts.

Making Music



Children don't need to understand musical notation for this activity. In fact, learning to read music is an exercise in pattern recognition itself. Start by explaining notes, making a link to keys on a piano. The numbers on the piano keys equate to the BYOB pitch number. The default number (60) represents the middle key on the piano. White piano keys are represented by the letters A to G, which repeat in sets of 8 (or octaves). Black notes are introduced later.

A musical score is a set of instructions to play sequences of notes. Having worked out the notes, convert to pitches (numbers) used in the 'Play Note' command. We are translating one language (a musical score) to a sequence of computer instructions. Rules for converting have to be expressed unambiguously (an example of an algorithm). The musical score itself is an algorithm – an unambiguous sequence of steps to produce a tune.

The different look of a musical note indicates the length of time it plays. All notes initially are 1 beat (a crotchet) apart from two examples which are two beats (a minim). If the sequence sounds too slow, do not alter the number of beats, but change the tempo. Armed with this knowledge, pupils script a whole line of a nursery rhyme - before completing the whole sequence.

The words of the rhyme should make it obvious that part of it is repeated. Can pupils spot the repeating musical sequence displayed? This is a good example of pattern recognition. If we can spot this, we can reuse the original sequence (initially just by copying it). Later we can look at better ways to reuse the code. It should become obvious that a long script makes it very difficult to spot mistakes. Far better to break the song down into smaller sections – using decomposition to simplify the task. This also introduces the use of message broadcasts to control program flow. Having refactored code into manageable chunks, it is a small conceptual step to encapsulate it in a 'songline' block of its own. Once defined, the rhyme can now be built from a higher level sequence of songline blocks. Hiding details in a new block (procedure) illustrates abstraction: the complexity is hidden, making the structure clearer. In this sense procedural abstraction and decomposition can be viewed as two sides of the same coin. Once sequences are defined as procedures, we can encapsulate those into higher level blocks too. There is no end to this process.

Complex code is built by decomposing a problem into successively smaller parts until we get to a manageable task. This can be coded, tested and, once working, the detail abstracted away in procedure, or block. Small blocks can be assembled to produce more complex sequences, which can then be tested and further encapsulated. Decomposition and abstraction lie at the heart of how we manage the inherent complexity of difficult problems.

To play our tune, we only need to 'call' or use one block, 'TwinkleRhyme', but what's the point of developing such a high level block? With such a high level block, we can tackle more complex problems, such as playing a round. There is an activity sheet for children and it is best to set this as a challenge to grapple with. The presentation outlines the further steps required to create a round. Once successful, a follow up activity sheet introduces 4 more challenging rounds to create.

Light-Bot is a wonderful game to reinforce defining functions and calling them multiple times. The Hour of Code sampler (lightbot.com/hoc2014.html) has simple challenges to allow children to build basic sequences from 5 command blocks (the last is a jump command). Very quickly, the number of spaces in the Main program are not enough to solve the challenge, and a procedure call is introduced. It gets progressively more challenging, but one word of warning. At the higher level, procedure calls are used 'recursively'. This means a procedure includes a call to itself in the definition, in order to create loops. An example is shown that would result in an infinite loop. Recursion is a tricky concept and when to introduce the idea is a source of pedagogical debate.