Teach Computing Science

A Guide for Early Years and Primary Practitioners

Teach Computing Science

A Guide for Early Years and Primary Practitioners

Kate Farrell, Computing at School Scotland
Professor Judy Robertson, University of Edinburgh
Professor Quintin Cutts, University of Glasgow
Professor Richard Connor, University of Strathclyde

With funding from

Introduction

The Scottish curriculum consists of individual learning outcomes, called Experiences and Outcomes (Es and Os), which are grouped into curricular areas. Recently, Significant Aspects of Learning (SALs) have been introduced as overarching themes across groups of Es and Os, and Benchmarks have been added as examples of typical activities.

This guide introduces and explains the Computing Science (CS) SALs and the updated experiences and outcomes and Benchmarks. It provides an exemplification guide and resources for use in Early and Primary years. It is the result of four years of work drawing on:

- Research literature in CS education
- A range of international curriculum efforts
- Experience of best-practice CS pedagogy in Scottish primary and secondary contexts
- Teaching resources from across the world

The authors are all practising CS educators, bringing experience of teacher education, CS education research, resource creation, as well as deep knowledge of the discipline of computing science. They are keenly aware of the challenges involved in CS teaching.

An innovative contribution of the framework (compared to curricular frameworks worldwide) is the organisation of core computational thinking (CT) concepts according to three SALs. The first SAL introduces learners to core concepts in CS. The second SAL introduces learners to how tools and languages use those concepts. The third SAL sees learners apply their learning from the first two SALs by creating solutions.

This structuring highlights key learning steps that are often overlooked by teachers, but that are essential if all learners are to succeed. We recognise that CS-specific knowledge and skills are necessary before learners can successfully solve problems.

In particular, the second SAL concentrates on *understanding* computer languages. Research and experience is showing that this understanding is essential before learners can successfully solve problems using those languages.

Particular points for readers to note:

- While a complete BGE progression framework is presented, only Early, First and Second levels are covered in detail here. We have focussed on these levels initially as there is no guidance currently available for primary teachers, and hence it is crucial to fill this gap. In due course, after further consultation with secondary teachers, we will gradually expand the Third and Fourth level to be suitable for learners who have worked within this framework at primary school.
- Readers will come across a few items within this computing progression framework that are common to other curricular areas. This is intentional. The important aspect here is to appreciate the whole developmental sequence required to understand CS and develop CT skills.
- We recognise that this is a work in progress. While we have based this on the best thinking currently available, we would expect this document to evolve and expand over time as we observe and reflect on teaching in action.

Based on our wide reading and experience, embodied in the framework, we are confident that ***all*** pupils can learn to think computationally. We think it is essential that this learning is started as early as possible, boosting equality of opportunity across both gender and background. We look forward to hearing from practitioners working with this framework, giving us advice on what works well and what is less effective - and - we wish you the very best in bringing this essential and exciting subject to your pupils.

Significant Aspects of Learning

Significant Aspects of Learning for Computing Science

Significant Aspects of Learning (SALs) are overarching themes across groups of individual learning outcomes, called Experiences and Outcomes (Es and Os). There are three Significant Aspects of Learning for Computing Science in the Scottish curriculum. These are as follows:

SAL 1: **Understanding the world through computational thinking**
This SAL is about: Theory Understanding the world through computational thinking and knowledge of core computing science concepts is necessary in order to later apply that knowledge using languages and technology.

SAL 2: **Understanding and analysing computing technology**
This SAL is about: Languages and Tools Understanding of computing technology and the programming languages that control them is essential before designing and building using these tools.

SAL 3: **Designing, building and testing computing solutions**
This SAL is about: Creating Use conceptual and technological knowledge to design, build and test.

The SALs for CS are structured to assist teachers in recognising key developmental stages in learning about computing concepts. This enables teachers to identify and correct learner misconceptions early on - something which is notoriously difficult to do when CS education is centred on creation.

The SALs don't focus directly on bits of computing kit or developing cool programs, unlike more traditional CS approaches. Instead, they show how, before we can get a computer to do anything useful for us (SAL 3), we need to understand precisely how computers are told to do anything at all (SAL 2) - and to understand that, we need to know what kinds of tasks computers can carry out (SAL 1).

The Significant Aspects of Learning for Computing Science build upon each other. The conceptual knowledge gained when working towards the first SAL, '**understanding the world through computational thinking**', is required to then **understand computing languages and technologies** in the second SAL, before we can then '**design, build and test computing solutions**' in the final SAL using those technologies.

It is expected that learners will be able to understand more complicated concepts in the first SAL than they are capable of reading or writing themselves in the second and third SALs. Similarly, their comprehension of representations and code written by someone else will likely outstrip their ability to write similarly complex code.

Most importantly, this does not mean that learners must gain an understanding of **all** of the concepts (SAL 1), languages and tools (SAL 2) before going on to develop and build computing solutions (SAL 3). As the SALS complement each other, it is expected that more than one SAL could be covered in a single lesson. It is a spiral curriculum, where the learners will revisit concepts at increasing depth as they work through the Levels.

The important thing is to ensure that learners are not expected to write correct programs (SAL 3) without knowledge and understanding of the underlying concepts (SAL 1) or being able to accurately read and understand programs in that language (SAL 2).

Definitions of the Significant Aspects of Learning in Computing Science

Understanding the world through computational thinking	The first Significant Aspect of Learning looks at the underlying theory in the academic discipline of Computing Science. Theoretical concepts of Computing Science include the characteristics of information processes, identifying information, classifying and seeing patterns. This strand is about understanding the nature and characteristics of **processes** and **information**. These can be taught through 'unplugged' activities (fun active learning tasks related to Computing Science topics but carried out without a computer) and with structured discussions with learners. There is a focus on recognising computational thinking when it is applied in the real world such as in school rules, finding the shortest or fastest route between school and home, or the way objects are stored in collections. Learners will be able to identify steps and patterns in a **process**, for example seeing repeated steps in a dance or lines of a song. In later stages, learners will begin to reason about properties of processes, for example considering whether tasks could be carried out at the same time, whether the output of a process is predictable, and how to compare the efficiency of two processes. Learners will identify **information**, classify it and see patterns. For example, learners might classify and group objects where there is a clear distinction between types or where objects might belong to more than one category.

Understanding and analysing computing technology	This significant aspect of learning aims to give learners insight into the hidden mechanisms of computers and the programs that run on them. It explores the different kinds of language, graphical and textual, used to represent processes and information. Some of these representations are used by people and others by machines. For example, a set of instructions could be represented as a verbal description, a sequence of blocks in a visual programming language such as Scratch, or as a series of 1s and 0s in binary. In this SAL learners will learn how to 'read' program code (before writing it in the next SAL) and describe its behaviour in terms of the **processes** they have learned about in the first SAL, processes that will be carried out by the underlying machinery when the program runs. For example, learners could read a section of code and predict what will happen when it runs or if lines of code change order. Learners will learn and explore different representations of **information** and how these are stored and manipulated in the computing system under study. A programming language defines a computing system. This SAL also covers how other computer systems work, including the components of an individual computer, configurations of networked computers and software systems such as a search engine.

Designing, building and testing computing solutions	The third SAL is about taking the concepts and understanding from the first two SALs and applying them. Learners will create solutions, perhaps by designing, building and testing solutions on a computer or by writing a computational process down on paper. In doing so, they will learn about modelling process and information from the real world in programs, and what makes a good model to represent or solve a particular problem. Learners will create representations of **information**. For example, learners could make lists, tables, family trees, Venn diagrams and data models to capture key information from the problems they are working on. Learners will use their skills in language to create descriptions of **processes** that can be used by other people. For example, a computer program is a great way to describe a process. Learners will understand how to read, write and translate between different representations such as between English statements, planning representations and actual computer code. For example, developing skills in writing code could be scaffolded by studying worked examples or by giving learners jumbled lines of code and asking them to put the lines into an order that will give the correct outcome. Although solutions can be created in a many different ways, it is expected that all learners will experience creating a variety of solution on computers. This will show learners that the computer will implement exactly what they have written and not what they intended, as well as giving them practice in debugging.

Themes across the Significant Aspects of Learning:
Information and Process

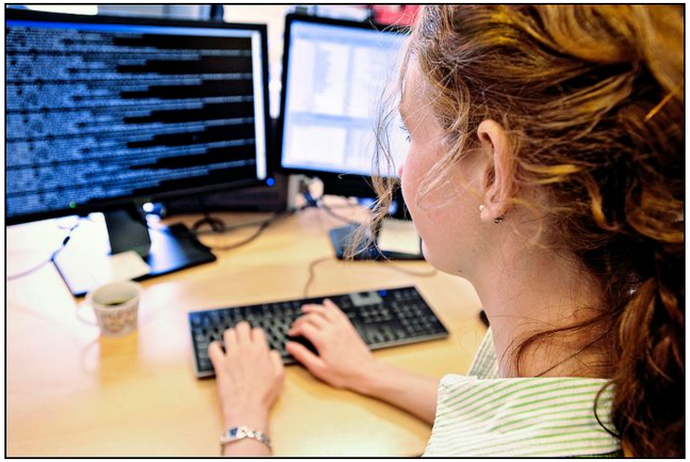
Running through the three SALs are the concepts of **processes** and **information**. Computers are just machines that carry out well-defined processes that manipulate information. Imagine a child carrying out a long multiplication sum on paper (old-school!). She is carrying out a process (writing down numbers and lines, repeated additions) that involves information (numbers, their positioning on the paper, carry overs). Although it may seem surprising, at the heart of all the amazing digital technology around us - e.g. computer games, self-driving cars, immersive 3D worlds, video-conferencing, on-line banking and shopping - are similar processes that manipulate information. So - to understand CS and to think computationally, we need to develop a steadily deepening understanding about processes and information. Interestingly, we often don't need computers for this! This is all explored in the first SAL.

A computing system does not spontaneously decide what process to carry out, or what information to manipulate. It is told precisely what to do via a set of instructions held in a computer program. These instructions are written in a programming language. Such languages are not at all like the sort of language we encounter every day - our spoken or signed natural language. Appreciating the difference is *very* important. Also, it is crucial to take time to learn the language thoroughly enough to be able to read and understand exactly what programs written in the language *mean*. Understanding the program enables us to be able to say, ahead of time, what process the computer will carry out and what information it will manipulate as it follows the instructions in the program. To do this, we must understand how each instruction affects the computer.

This is harder than it may seem, because the internal operation of the computer is largely hidden from us. We can't see what is going on so we must rely on complex mental models to understand this. If these mental models are incorrect or missing then we might think a computer is magic (or out to get us!) rather than simply following a process that is described by its instructions. The good news is that despite the huge number of different computing devices and programming languages, they are remarkably similar, and hence learning a set of core principles and skills will take us a long way. This mix of understanding languages, representations and how they influence the machines they run on is captured in the second SAL.

The third SAL is typically the focus of computing courses - taking a problem or task and writing a program so that a computer can solve the problem or carry out the task. It's often thought to be the exciting bit (although we'd argue the other SALs can be just as much fun!) SAL 3 covers both the creation of programs to solve problems, and also how to determine whether they are correct and how to fix them if they're not.

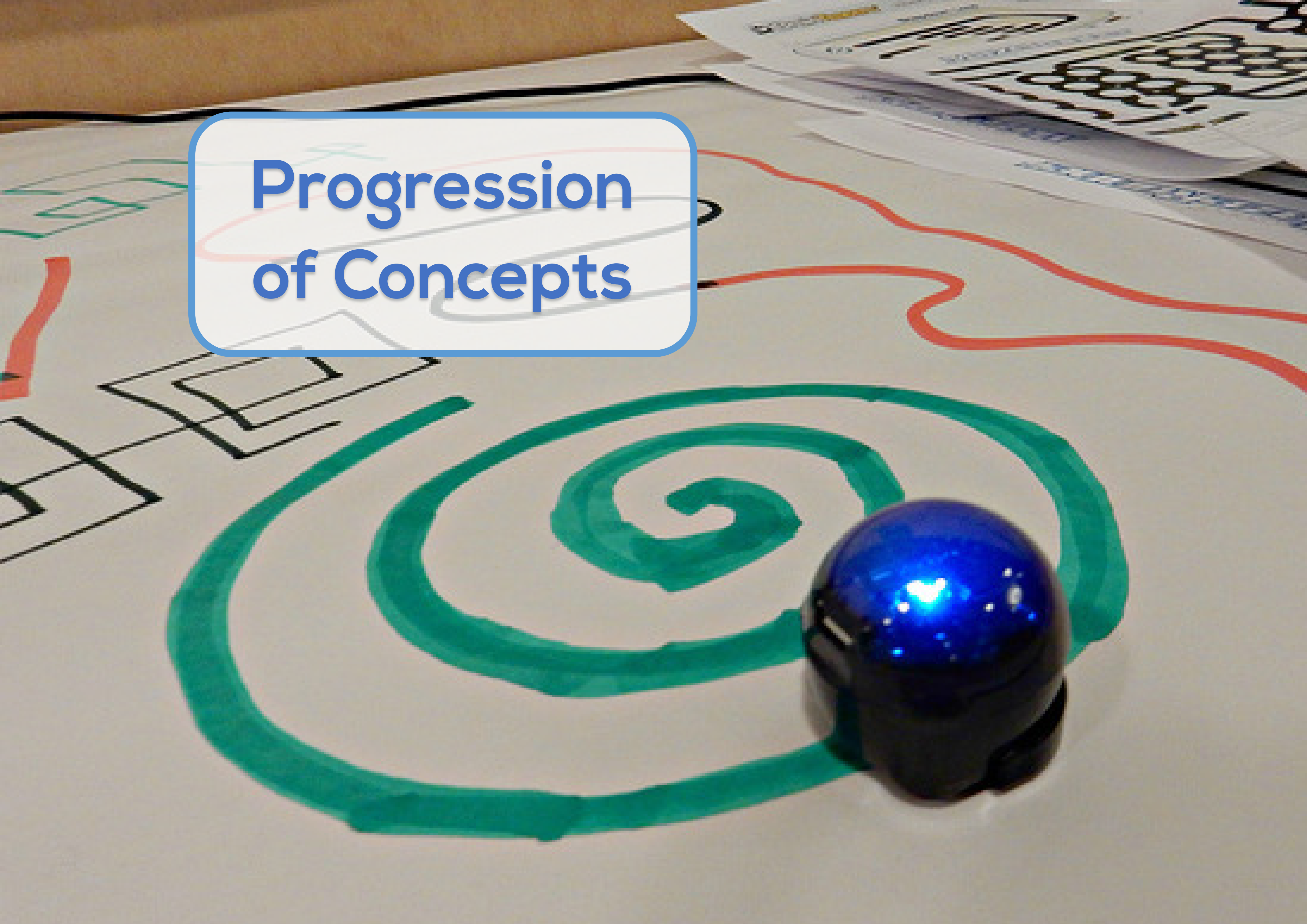
It might seem unusual for a computing curriculum to have programming included only in the third strand. However, this is a strength of this framework. How can we hope to instruct a computer to do what we want if we don't understand the fundamental nature of what its operation involves - processes and information (SAL 1)? And equally, we're unlikely to be successful if we don't thoroughly understand the means of communicating our instructions to the computer - the programming language (SAL 2). It would be like trying to write a car repair manual without understanding anything about cars and engineering, nor about the English language and diagrams!



In all the above, we've written about computers and programming languages. But the scope here is much broader. The same set of core principles and skills applies to databases, web systems, digital networks, mobile systems and so on. They all use processes and information of varying kinds. They all have languages of instruction. And we take problems or tasks and write solutions that will operate on these systems using similar approaches and techniques.

Finally, it is clear that many of the concepts and skills learned here are of value more broadly than computing science and are translatable to other contexts. Modern life is often complex and involves processes and information even where computers are not directly involved. Computational Thinking, which this framework is designed to develop, can help in all kinds of situations at home, in school and in the workplace.

Progression of Concepts

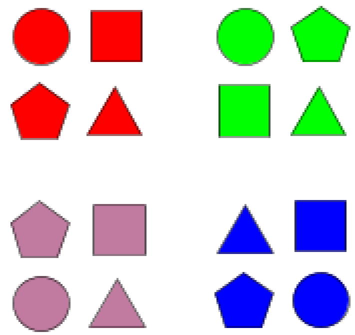


Core Computing Science Concepts across the SALs

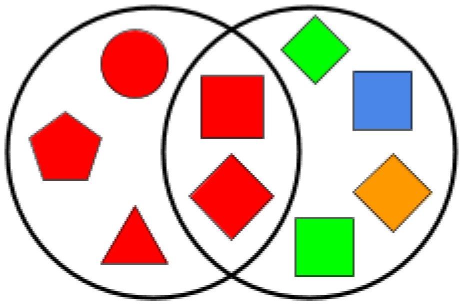
As noted earlier, the SALs help teachers to structure the learning of computing concepts. This table outlines the range of individual computing concepts that have been incorporated into the progression framework. You will see elements of each across the three SALs.

Concepts	Early Level	First Level	Second Level
Structuring Processes	Sequence (of movements)	Simple sequences Selection - sequences with conditional statements	Single or parallel sequences Variables
Patterns in Processes	Spotting patterns in processes	Fixed repetition - Identifying patterns that repeat a predetermined number of times	Fixed or conditional repetition - processes that repeat a fixed number of times or until a condition is met
Structuring and manipulating information	Basic sorting - classifying of objects according to characteristics	Grouping and ordering of information collected from objects Using logic (AND, OR, NOT) to sort objects depending on different conditions	Following sorting algorithms Structuring and manipulating information, such as family trees
Computing Systems	Computers follow instructions	Computers take in inputs, process them, store information, and then output the results	Computers can communicate over networks

Progression of Information concepts in SAL 1: Understanding the world through computational thinking



Early level:
Learners can classify objects and group them into simple categories. For example, they can group toys based on type, colour or size. They can spot similarities and differences in objects and identify simple relationships between them. Learners can also identify patterns in objects and information



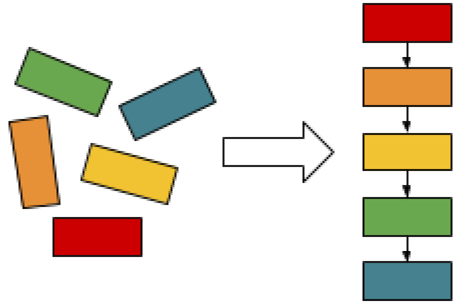
First level:
Learners can classify based on multiple categories. They can use a range of ways to collect information and can group it in a logical, organised way using their own and others' criteria.

Learners can classify and make decisions based on logical thinking. Logical decisions include AND (collecting objects that are red AND square), OR (choose the Water OR Ice Pokemon creatures) and NOT (put away your jotters but NOT your Maths jotter)

Second level:
Learners will be aware that information can be sorted, and be able to perform a simple sorting algorithm on real world objects.

They can structure related items of information, for example arranging family members into a family tree, or classifying animals according to species.

Progression of Process concepts in SAL 1: Understanding the world through computational thinking



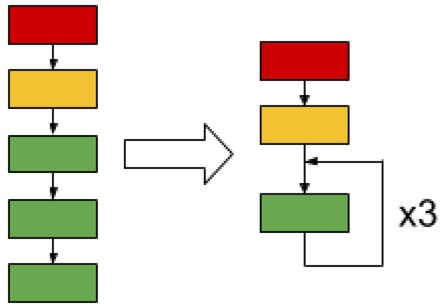
Early level:

Learners are able to identify the beginning and end of a process and the steps in between. This might be demonstrated by a learner programming a toy robot with a set of instructions or by giving someone directions to a familiar place.

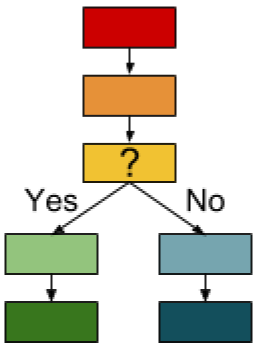
First level:

Learners will be able to demonstrate knowledge of processes by being able to follow instructions in a recipe or understanding their role in tidying the classroom or in Scottish country dancing. Learners can construct sequences of steps such as pirate treasure maps, directions to secret locations, instructions on how to make a jam sandwich.

Learners should be able to describe the effect of each step in a sequence. They will be able to look at a set of steps and predict what the outcome will be, for example identifying where they will end up in their school when they follow a set of directions



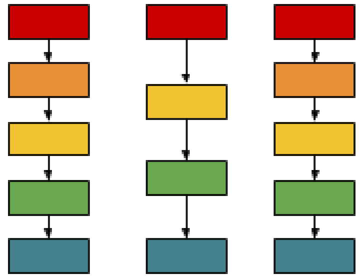
Learners can identify similarities and differences in a set of steps in a process. They can spot patterns that are identical, repeating or where steps are similar. Learners can describe how patterns are similar (such as ascending or descending numbers in songs or games).

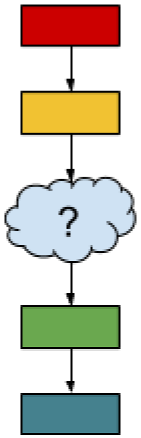


Learners understand how decisions (for example a test with a yes/no answer) can be used for selection, to introduce choice in processes. Learners can make decisions based on logical thinking (for example IF your painting is dry THEN put it in your tray ELSE put it on the table.)

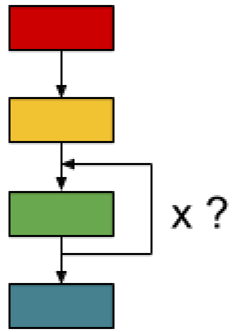
Second level:

Learners can identify when a process is a single sequence or consists of multiple parallel steps, such as team relay races and balloon passing party games. Later, in SALs 2 and 3, learners will be able to identify parallel processes such as Scratch programs with multiple ‘sprites’ each following their own set of steps at the same time)





They can predict the outcome of a process or identify when a process is non-predictable (e.g it has a random element such as board games with a spinners or dice.)



Learners can identify when a repeated set of steps is fixed (it loops a known number of times) or conditional (it loops until a condition is met).

Learners can see patterns in problem solving and identify a solution that has been used previously. For example, when creating a set of instructions on how to get to the head teacher, they can reuse instructions on how to get to the school office, or instructions on how to cook pasta can be adapted for boiling rice. Later, if learners are creating games in SAL 3, they can reuse sections of code for different purposes, such as setting up a lap timer or controlling the character using arrow keys.

Learners will evaluate different solutions to a problem and evaluate them in terms of efficiency (smallest number of steps) and speed.

Progression of Languages in SAL 2: Understanding and Analysing Computing Technology

Early level:

When using floor turtles or simple robots, learners will start to understand the relationship between the symbols used to create instructions, and the behaviour of the device as it follows the instructions.

The important cognitive step is seeing the difference between pressing a button to immediately move a robot, to understanding an arrow button as a command which will cause a defined behaviour at some point in the future.

A learner should be able to understand a sequence of commands using a simple symbolic language such as arrows drawn on paper. They should be able to predict what the robot or person will do when it is presented with a sequence of instructions (as well as learn to debug their thinking if their prediction is proved false.)

First level:

The benefit of using an **icon-based block environment**, like ScratchJr or The Foos, is that learners without the literacy skills to read textual instructions on blocks in Scratch can still explore the computing concepts, just using pictorial icons instead of words (or colour-based instructions, in the case of Ozobot.)

Learners should become confident about understanding the precise *meaning* of each individual block - that is, for example, the effect that instruction has on a visual display when the program is run. This understanding of meaning should also incorporate the concept of running a whole program, with instructions being performed in order according to the layout of the program's blocks. This includes understanding how each instruction block cumulatively affects the world it operates on, for example, how the visual display is updated.

Second level:

Using a **block-based development environment** like Scratch, learners will be able to explain the meaning of more complex programs that include selection and repetition blocks. They understand that variables can change as the program runs through each instruction block. They will be able to predict what a complete program will do when it runs.

Progression of Languages in SAL 3: Designing, building and testing computing solutions

Early level:

Learners start to understand how to model behaviour in a robot or a computer character using a simple sequence of commands. For example, a floor robot can model the behaviour of moving towards a treasure box with a series of instructions such as "turn left", "go forwards" and so on.

More generally, a learner should be able to choose a destination for the robot, design a sequence of instructions that will cause the robot to move to that destination and, finally, enter that sequence of instructions and test whether they achieve the desired effect. If this end state is not a state that was planned, learners should identify and correct errors in their set of instructions.

Of course, designing and building computing solutions cannot be achieved without first understanding both the fundamental nature of step by step processes (SAL1) and the effect on the robot caused by running each instruction in the sequence (SAL2). While all three of these aspects may be very obvious in simple examples, it is essential from the beginning to unpick these before the learners progress onto more complex contexts.

First level:

At this level, languages and systems such as ScratchJr are used to design programs which fulfil more sophisticated and more abstract tasks than just movement commands. However, the essence is still using computing language skills (gained in SAL2) to achieve the design of abstract processes (SAL1).

Second level:

At this level, learners should start to understand that there are many ways of achieving the same outcomes, and that some of these are preferable to others. They should have encountered this in previous SALs through exploring concepts of efficiency and speed.

Learners start to understand the relationship between the meaning of programming constructs such as conditions and repetition, and the ways in which these can be used to achieve desired behaviour in a running program.

Experiences and Outcomes

SAL 1: Understanding the world through computational thinking

Experiences and Outcomes				
Early Level	**First Level**	**Second Level**	**Third Level**	**Fourth Level**
I can explore computational thinking processes involved in a variety of everyday tasks and can identify patterns in objects or information. **TCH 0-13a**	I can explore and comment on processes in the world around me making use of core computational thinking concepts and can organise information in a logical way. **TCH 1-13a**	I understand the operation of a process and its outcome. I can structure related items of information. **TCH 2-13a**	I can describe different fundamental information processes and how they communicate, and can identify their use in solving different problems **TCH 3-13a** I am developing my understanding of information and can use an information model to describe particular aspects of a real world system **TCH 3-13b**	I can describe in detail the processes used in real world solutions, compare these processes against alternative solutions and justify which is the most appropriate **TCH 4-13a** I can informally compare algorithms for correctness and efficiency **TCH 4-13b**
Benchmarks				
• Identifies and sequences the main steps in an everyday task to create instructions / an algorithm, for example, washing hands • Classifies objects and groups them into simple categories (MNU 0-20a, MNU 0-20b, MNU 0-20c), for example, groups toy bricks according to colour • Identifies patterns, similarities and differences in objects or information such as colour, size and temperature and simple relationships between them (MNU 0-13a)	• Follows sequences of instructions/algorithms from everyday situations, for example, recipes or directions, including those with selection and repetition • Identifies steps in a process and describes precisely the effect of each step • Makes decisions based on logical thinking including IF, AND, OR and NOT, for example, collecting balls in the gym hall but NOT basketballs, line up if you are left-handed OR have green eyes • Collects, groups and orders information in a logical, organised way using my own and others' criteria (MNU 1-20a and b)	• Compares activities consisting of a single sequence of steps with those consisting of multiple parallel steps, for example, making tomato sauce and cooking pasta to be served at the same time • Identifies algorithms / instructions that include repeated groups of instructions a fixed number of times and/or loops until a condition is met • Identifies when a process is not predictable because it has a random element, for example, a board game which uses dice • Structures related items of information, for example, a family tree (MNU 2-20b) • Uses a recognised set of instructions / an algorithm to sort real worlds objects, for example, books in a library or trading cards	• Recognises and describes information systems with communicating processes which occur in the world around me • Explains the difference between parallel processes and those that communicate with each other • Demonstrates an understanding of the basic principles of compression and encryption of information • Identifies a set of characteristics describing a collection of related items that enable each item to be individually identified • Identifies the use of common algorithms such as sorting and searching as part of larger processes.	• Identifies the transfer of information through complex systems involving both computers and physical artefacts, for example, airline check-in, parcel tracking and delivery. • Describes instances of human decision making as an information process, for example, deciding which check-out queue to pick, which route to take to school, how to prepare family dinner / a school event • Compares alternative algorithms for the same problem and understands that there are different ways of defining "better" solutions depending on the problem context for example, is speed or space more valuable in this context?

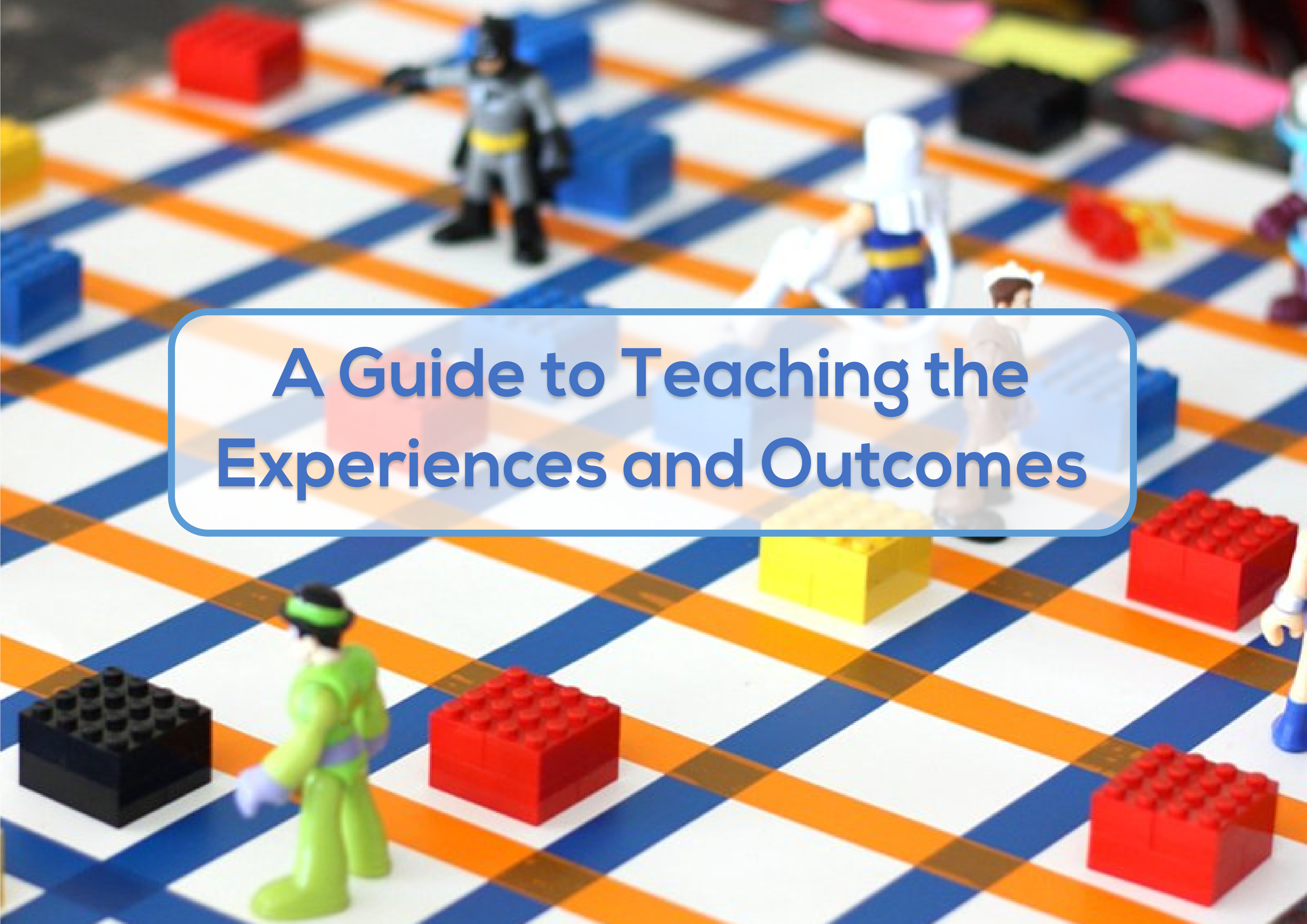
SAL 2: Understanding and Analysing Computing Technology

Experiences and Outcomes				
Early Level	**First Level**	**Second Level**	**Third Level**	**Fourth Level**
I understand that sequences of instructions are used to control computing technology **TCH 0-14a** I can experiment with and identify uses of a range of computing technology in the world around me. **TCH 0-14b**	I understand the instructions of a visual programming language and can predict the outcome of a program written using the language. **TCH 1-14a** I can understand how computers process information. **TCH 1-14b**	I can explain core programming language concepts in appropriate technical language **TCH 2-14a** I understand how information is stored and how key components of computing technology connect and interact through networks. **TCH 2-14b**	I understand language constructs for representing structured information **TCH 3-14a** I can describe the structure and operation of computing systems which have multiple software and hardware levels that interact with each other. **TCH 3-14b**	I understand constructs and data structures in a textual programming language **TCH 4-14a** I can explain the overall operation and architecture of a digitally created solution **TCH 4-14b** I understand the relationship between high level language and the operation of computer **TCH 4-1c**
Benchmarks				
• Demonstrates an understanding of how symbols can represent process and information • Predicts what a device or person will do when presented with a sequence of instructions for example, arrows drawn on paper • Identifies computing devices in the world (including those hidden in appliances and objects such as automatic doors)	• Demonstrates an understanding of the meaning of individual instructions when using a visual programming language (including sequences, fixed repetition and selection) • Explains and predicts what a program in a visual programming language will do when it runs for example, what audio, visual or movement effect will result • Demonstrates an understanding that computers take information as input, process and store that information, and output the results.	• Explains the meaning of individual instructions (including variables and conditional repetition) in a visual programming language • Predicts what a complete program in a visual programming language will do when it runs, including how the properties of objects for example, position, direction and appearance, change as the program runs through each instruction • Explains and predicts how parallel activities interact • Demonstrates an understanding that all computer data is represented in binary, for example, numbers, text, black and white graphics. • Describes the purpose of the processor, memory and storage and the relationship between them • Demonstrates an understanding of how networks are connected and used to communicate and share information, for example, the internet	• Understands that the same information could be represented in more than one representational system • Understands that different information could be represented in exactly the same representation • Demonstrates an understanding of structured information in programs, databases or webpages • Describes the effect of markup language on the appearance of a webpage, and understands that this may be different on different devices • Demonstrates an understanding of the von Neumann architecture and how machine code instructions are stored and executed within a computer system • Reads and explains code extracts including those with variables and data structures • Demonstrate an understanding of how computers communicate and share information over networks including the concepts of sender, receiver, address and packets. • Understands simple compression and encryption techniques used in computing technology	• Understands basic control constructs such as sequence, selection repetition, variables and numerical calculations in a textual language • Demonstrates an understanding of how visual instructions and textual instructions for the same construct are related • Identifies and explains syntax errors in a program written in a textual language • Demonstrates an understanding of representations of data structures in a textual language • Demonstrates an understanding of how computers represent and manipulate information in a range of formats • Demonstrates an understanding of program plans expressed in accepted design representations, for example, pseudocode, storyboarding, structure diagram, data flow diagram, flow chart • Demonstrates an understanding of the underlying technical concepts of some specific facets of modern complex technologies, for example, on line payment systems and SATNAV • Demonstrates an understanding that computers translate information processes between different levels of abstraction

SAL 3: Designing, building and testing computing solutions

Experiences and Outcomes				
Early Level	**First Level**	**Second Level**	**Third Level**	**Fourth Level**
I can develop a sequence of instructions and run them using programmable devices or equivalent **TCH 0-15a**	I can demonstrate a range of basic problem solving skills by building simple programs to carry out a given task, using an appropriate language. **TCH 1-15a**	I can create, develop and evaluate computing solutions in response to a design challenge. **TCH 2-15a**	I can select appropriate development tools to design, build, evaluate and refine computing solutions based on requirements. **TCH 3-15a**	I can select appropriate development tools to design, build, evaluate and refine computing solutions to process and present information whilst making reasoned arguments to justify my decisions. **TCH 4-15a**
Benchmarks				
• Designs a simple sequence of instructions/algorithm for programmable device to carry out a task for example, directional instructions: forwards/backwards • Identifies and corrects errors in a set of instructions	• Simplifies problems by breaking them down into smaller more manageable parts • Constructs a sequence of instructions to solve a task, explaining the expected output from each step and how each contributes towards solving the task • Creates programs to carry out activities (using selection and fixed repetition) in a visual programming language • Identifies when a program does not do what was intended and can correct errors/bugs • Evaluates solutions/programs and suggests improvements	• Creates programs in a visual programming language including variables and conditional repetition • Identifies patterns in problem solving and reuses aspects of previous solutions appropriately, for example, reuse code for a timer, score counter or controlling arrow keys • Identifies any mismatches between the task description and the programmed solution, and indicates how to fix them	• Designs and builds a program using a visual language combining constructs and using multiple variables • Represents and manipulates structured information in programs or databases, for example, works with a list data structure in a visual language or a flat file database • Interprets a problem statement and identifies processes and information to create a physical computing and/or software solution • Can find and correct errors in program logic • Groups related instructions into named subprograms (in a visual language) • Writes code in which there is communication between parallel processes (in a visual language) • Writes code which receives and responds to real world inputs (in a visual language) • Designs and builds web pages using appropriate mark-up languages	• Analyses problem specifications across a range of contexts, identifying key requirements • Writes a program in a textual language which uses variables and constructs such as sequence, selection and repetition • Creates a design using accepted design methodologies for example, pseudocode, storyboarding, structure diagram, data flow diagram, flow chart • Develops a relational database to represent structured information • Debugs code and can distinguish between the nature of identified errors e.g. syntax and logic • Writes test and evaluation reports • Can make use of logical operators – AND, OR, NOT • Writes a program in a textual language which uses variables within instructions instead of specific values where appropriate • Designs appropriate data structures to represent information in a textual language • Selects an appropriate platform on which to develop a physical and/or software solution from a requirements specification • Compares common algorithms for example, those for sorting and searching, and justify which would be most appropriate for a given problem • Designs and builds web pages which include interactivity.

A Guide to Teaching the Experiences and Outcomes



SAL 1 Early Level	Understanding the world through computational thinking
Outcome:	**Benchmarks:**
I can explore computational thinking processes involved in a variety of everyday tasks and can identify patterns in objects or information. **TCH 0-13a**	• Identifies and sequences the main steps in an everyday task to create instructions / an algorithm, for example, washing hands • Classifies objects and groups them into simple categories (links to MNU 0-20a, MNU 0-20b, MNU 0-20c), for example, groups toy bricks according to colour • Identifies patterns, similarities and differences in objects or information such as colour, size and temperature and simple relationships between them (links to MNU 0-13a)

More information:

Process - sequences of steps and changing states
Through awareness of everyday tasks and objects, learners are able to identify the beginning, intermediate steps, and ending stages of a process. Learners will start to understand cause and effect as they see steps changing an object from a starting state to an end state.

Information - Classifying objects
Learners can identify the basic features of objects, and classifying them according to different attributes such as colour, size, and temperature. They can spot similarities and differences in objects and identify simple relationships between them. Learners can also identify patterns in objects and information.

What this learning may look like:

Information - Classifying objects
Learners will gain experience identifying basic features of objects and classifying them according to different attributes such as colour or size. Objects can then be compared according to an attribute - bigger, softer, noisier - and then sorted. A good example of this would be asking learners to sort a pile of Lego into an order they choose, perhaps tidying it into a organiser storage, drawers or bags. Will learners choose to organise by colour or size and what will their exceptions and special cases be? It is useful for the learners to realise that objects have more than one attribute and so there are different valid ways to categorise and sort a collection of objects. Ask learners to think about the relationship between groups of objects, such as the order you sort the classified group. For example do the Adventure books go next to the Science Fiction books or the Action story books on the shelf?

Process - sequences of steps and changing states
As learners experiment they will be able to identify and sequence steps that give a desired end state. Marble runs or domino runs are great examples of this, where the change of state is both obvious to see and fun too! There are a number of videos of amazing Rube Goldberg machines too, such as the music video for OK Go This Too Shall Pass. Learners can experiment with changing state in different ways. They can pour water down drain pipes then change the direction of the water by moving the drain pipes.

SAL 2 Early Level	Understanding and analysing computing technology
Outcome:	**Benchmarks:**
I understand that sequences of instructions are used to control computing technology. **TCH 0-14a** I can experiment with and identify uses of a range of computing technology in the world around me. **TCH 0-14b**	● Demonstrates an understanding of how symbols can represent process and information ● Predicts what a device or person will do when presented with a sequence of instructions for example, arrows drawn on paper ● Identifies computing devices in the world (including those hidden in appliances and objects such as automatic doors)

More Information:

Process: Learners can read and understand various representations of simple processes (e.g picture cards with arrows) when reading from different representations such as blocks in a visual language or a flow diagram.

Information: They can understand simple pictorial, or physical representations of real-world information and make deductions about the real world from such representations.

Computing technology: Learners appreciate the world around them contains computing devices that perform useful activities.

What this learning may look like:

Learners can use different representations, such as looking at a sequence of cards with arrows printed out on them. They can then predict what toy robot would do if it was given those instructions with a particular starting point on a map. This could be simulated with a person playing the role of a robot if you don't have a robot. Learners can appreciate pictorial instructions, such as instructions for building Lego models.

Learners can understand simple ways of displaying information and use this to reason about the world. For example, if there are jars to represent different primary colours, each class member puts a coloured bead into one of the jars, and learners identify which was the most popular colour by observing which jar was fullest.

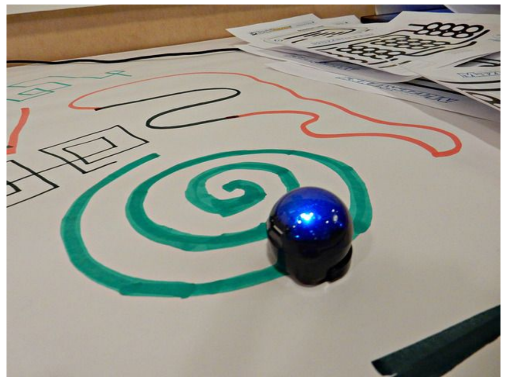
Learners are explicitly introduced to an everyday object with embedded computing technology - for example learning about an automatic door and exploring how the sensors work by sneaking up to it.   They can pick between pictures of everyday objects, identifying those making use of computing technology. They could also do this on school excursions.

SAL 3 Early Level	Designing, building and testing computing solutions
Outcome:	**Benchmarks:**
I can develop a sequence of instructions and run them using programmable devices or equivalent. **TCH 0-15a**	● Designs a simple sequence of instructions/algorithm for programmable device to carry out a task for example, directional instructions: forwards/backwards ● Identifies and corrects errors in a set of instructions

More Information:

Learners can understand how a short sequence of precise instructions can be interpreted by a device to carry out a simple task, for example moving a robot or graphic turtle from one place to another. They can understand the difference between a “correct” and “incorrect” instruction sequence. Learners can identify and correct any errors in the sequences they create.

What this learning may look like:



At this stage, learning is focussed on giving simple instructions either to teachers, to other children, or to toy robots. Learners should be able to find and correct errors in instructions when the robot does not do as they expected.

Although Beebots have been a common sight in Primary schools for many years, the instructions they follow are not visible to learners. They are hidden away inside the robot. An alternative is to use Ozobot, which follows a line drawn on paper and does different things depending on the colours of the line. Ozobot’s instructions are encoded on paper, and learners can be asked to ‘read’ the instructions and predict what the robot will do. Ozobot can be used at Early, First and Second level.

There are many other toy robots available such as Sphero, SPRK, Ollie, or Dash and Dot. Many of the other new robots on the market are controlled by a block interface on a computer or tablet app, so learners can follow along as the robot carries them out. Some of these robots (such as the Sphero robot ranges including Ollie and SPRK) have many different ways to control them such as direct commands (like a remote control car), drawing lines (controlling the Sphero robot ranges using the Draw tool in the Lightning Lab app, ideal for Early level learners) or using block-based apps like Tynker (similar to Scratch, this would be more suitable for learners at First and Second level) or using Lightning Lab. Learners could also build tracks or mazes for a robot to negotiate.

For schools that can’t afford to buy robots, there are many tablet apps and online games available that allow learners to program a robot. For example, Beebot have a web-based emulator. There are also simple board games available, such as Bits and Bytes and Robot Turtle which teach young children the concepts of creating and following step by step instructions.

SAL 1 First Level	Understanding the world through computational thinking
Outcome:	**Benchmarks:**
I can explore and comment on processes in the world around me making use of core computational thinking concepts and can organise information in a logical way. **TCH 1-13a**	● Follows sequences of instructions/algorithms from everyday situations, for example, recipes or directions, including those with selection and repetition ● Identifies steps in a process and describes precisely the effect of each step ● Makes decisions based on logical thinking including IF, AND, OR and NOT, for example, collecting balls in the gym hall but NOT basketballs, line up if you are left-handed OR have green eyes ● Collects, groups and orders information in a logical, organised way using my own and others' criteria (MNU 1-20a and b)

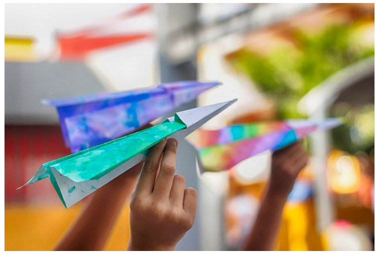
More information:

Process: Learners can identify and use (in simple English language) the control flow concepts of **sequence**, **selection** and **repetition**. They should be familiar with sequence from Early level, as a set of step by step instructions. Learners can understand simple sequences and correctly carry out a role assigned to them in a process such as a game, story or dance.

Information: Learners will begin to understand more complex logical constructs including AND, OR and NOT in order to group real world objects or follow verbal instructions requiring decisions to be made.

What this learning may look like:

Process: Learners will be able to demonstrate knowledge of processes by being able to follow instructions in a recipe or understanding their role in tidying the classroom or country dancing. Learners can follow **sequences** of steps such as directions on a pirate treasure maps and instructions to carry out an activity such as making a paper airplane. They can identify and describe the steps involved in games and movement dances such as the Macarena or Locomotion.



Learners can identify similarities and differences in a set of steps in a process. They can spot **repetition,** patterns that are identical or where steps are similar. Learners can describe how patterns are similar (such as ascending or descending numbers in songs or games).

Learners understand how decisions can be used to introduce **selection** between alternative processes. IF a condition is met THEN do something, ELSE do something different. Learners can explore selection statements through everyday situations such as learning to cross the road. IF the green man is lit THEN look both ways and cross the road ELSE wait patiently.

Information: Learners can follow **logical** instructions and make decisions involving AND, OR and NOT. It might be they are grouping real world objects (such as collecting all the Lego blocks but NOT the red ones, or collecting pencils OR pens in the classroom) or follow verbal instructions (Stand at the back if you are tall AND like singing).

SAL 2 First Level	Understanding and analysing computing technology
Outcome:	**Benchmarks:**
I understand the instructions of a visual programming language and can predict the outcome of a program written using the language. **TCH 1-14a** I can understand how computers process information. **TCH 1-14b**	• Demonstrates an understanding of the meaning of individual instructions when using a visual programming language (including sequences, fixed repetition and selection) • Explains and predicts what a program in a visual programming language will do when it runs for example, what audio, visual or movement effect will result • Demonstrates an understanding that computers take information as input, process and store that information, and output the results.

More Information:

Process: Learners can read, understand and explain representations of processes expressed in a programming language, with control flow elements of sequence, selection and fixed repetition. Learners understand how conditions can be used to decide between alternative sequences of steps.
Information: Learners can understand diagrams which illustrate key aspects of information (such as Venn and Carroll diagrams). Tables and diagrams are introduced as a way of representing information about collections of objects.
Computing technology: Learners know about input devices such as sensors, touch screens, keyboards and mice, output devices such as screens, speakers, and motors, and how these are connected to a processing unit. They can make links between information and process concepts and what is going on inside computing technology.

What this learning may look like:

Learners can "read" the blocks in an icon-based visual programming language such as ScratchJr to understand what a simple program will do. They can identify when there are mistakes in a program written in a simple visual language.

Learners can understand a range of diagram types which display information. They can look at a diagrams and charts, and answer questions based on the information presented graphically or numerically.

Learners can learn about different input devices (such as keyboards, mice, microphones) that send signals into computers; about how the computer processes the signals; and how the computer then sends signals to output devices (such as monitors, speakers) so that we can see/hear the result. Learners can then explore different computer systems such as games consoles or mobile phones to identify which input and output devices are used with that system. Learners may also tinker with the insides of an old computer if one is available.

Learners can make the link between the programming concepts they are learning and what is going on inside every-day devices. For example, a device playing a beeping sound is repeating the same action (playing a single beep and pausing) many times. A 'play pop sound' command in ScratchJr will cause the device to make a pop noise.

SAL 3 First Level	Designing, building and testing computing solutions
Outcome:	**Benchmarks:**
I can demonstrate a range of basic problem solving skills by building simple programs to carry out a given task, using an appropriate language. **TCH 1-15a**	• Simplifies problems by breaking them down into smaller more manageable parts • Constructs a sequence of instructions to solve a task, explaining the expected output from each step and how each contributes towards solving the task • Creates programs to carry out activities (using selection and fixed repetition) in a visual programming language • Identifies when a program does not do what was intended and can correct errors/bugs • Evaluates solutions/programs and suggests improvements

More Information:

Learners start to understand how they can use simple programming constructs, such as repetition, to achieve desired behaviour in a more concise manner. They are introduced to the idea that different instructions can be used to achieve the same behaviour, and that some are better than others.

What this learning may look like:

Learners will use a visual programming language to create simple programs and animations. An icon-based environment such as would be best, as they do not rely on good literacy skills. Hour of Code's has a course aimed at 4-6 year olds that would be ideal for playing on an interactive whiteboard.

For schools with access to tablets, many apps allow learners to program in a simple way. Apps such as ScratchJr, Cargobot, Robot School, Move the Turtle, The Foos, Kodable, and Daisy the Dinosaur are appropriate, particularly those which rely on pictures and symbols rather than text. To give children more interesting or complex problems to work on, they can be given a partly completed program and asked to write the code to extend one part of it.

Robots are an engaging way for learners to develop simple algorithms. Learners using Ozobot (see Early level) at First level would be expected to start using colour codes to program the robot. See the Early level notes on robots for more ideas.

Lego WeDo is a great way for learners to explore with creating programs in an icon-based environment. The software features two Lego characters who guide learners through different challenges. They build mechanical solutions using Lego pieces and develop an algorithm using the software.

There are also simple board games available, such as Bits and Bytes and Robot Turtle, which teach young children the concepts of creating and following step by step instructions. Learners could make their own programming board game.

SAL 1 Second Level	Understanding the world through computational thinking
Outcome:	**Benchmarks:**
I understand the operation of a process and its outcome. I can structure related items of information. TCH 2-13a	● Compares activities consisting of a single sequence of steps with those consisting of multiple parallel steps, for example, making tomato sauce and cooking pasta to be served at the same time ● Identifies algorithms/instructions that include repeated groups of instructions a fixed number of times and/or loops until a condition is met ● Identifies when a process is not predictable because it has a random element, for example, a board game which uses dice ● Structures related items of information, for example, a family tree (MNU 2- 20b) ● Uses a recognised set of instructions / an algorithm to sort real worlds objects, for example, books in a library or trading cards

What this learning may look like:

Process

Learners can identify and describe the properties of simple systems such as parallel processes. Learners could play a part in a physical process (such as in a playground game or dance) in which multiple activities are carried out at the same time (parallel). For example, teams of learners could race to complete a process (such as a balloon party game or tidying their workspace). Learners then discuss the effectiveness of different team strategies. Sometimes more than one type of activity might be carried out at the same time (with individuals or teams having different tasks to complete as part of a bigger goal, such as tidying the classroom at the end of the day). Learners can identify when two activities which happen at the same time interact with each other.

Through playing and analysing common games (such as Snakes and Ladders, or noughts and crosses) they can identify when it is possible to predict what will happen in a process and the circumstances under which it will end. This includes understanding about randomness, often appearing in games with dice.

Information

Learners will be aware that information can be sorted, and be able to perform a simple sorting algorithm on real world objects. They could sort books alphabetically or a deck of Pokemon cards by combat power. There are 'unplugged' activities listed for sorting. You could also watch a video of a sort algorithm being carried out (such as this [Lego version of the bubble sort] or a [kids song version] or watching [robots do quick sort].)

Learners could write down all the family members they know, drawing lines between them to represent family relationships, and then be shown how to redraw the information in a family tree format. Learners could attempt to form a similar structure with everybody they can think of in the school - pupils, teachers, admin staff, etc.

SAL 2 Second Level	Understanding and analysing computing technology
Outcome:	**Benchmarks:**
I can explain core programming language concepts in appropriate technical language. **TCH 2-14a** I understand how information is stored and how key components of computing technology connect and interact through networks. **TCH 2-14b**	● Explains the meaning of individual instructions (including variables and conditional repetition) in a visual programming language ● Predicts what a complete program in a visual programming language will do when it runs, including how the properties of objects for example, position, direction and appearance, change as the program runs through each instruction ● Explains and predicts how parallel activities interact ● Demonstrates an understanding that all computer data is represented in binary, for example, numbers, text, black and white graphics. ● Describes the purpose of the processor, memory and storage and the relationship between them ● Demonstrates an understanding of how networks are connected and used to communicate and share information, for example, the internet

More Information:

Process
Learners can read and understand representations of processes in a programming language, identifying examples of sequential, selective, repetitive and parallel control structures and uses of variables. They can pick out and describe the precise meaning of individual structures, as well as predict the outcome of programs using all of these types of structures expressed in a well-specified language. This might be a visual block language such as Scratch, Alice or Kodu, or a text-based language such as HTML.

Information
Learners understand how different types of information are stored by the computer, for example the basics of how text, sound and images are handled by the computer. They understand that digital representations are a lingua franca allowing widely different kinds of information to be represented in a uniform format that can be transmitted over networks and interpreted by a wide variety of different kinds of device, for example a video from your mobile phone can be emailed to a friend and displayed on their television with a sound track from another friend.

Computing technology
Learners understand that many modern computer applications are made up of many computers connected in a network. They understand the principles of how information can be transmitted between networked computers, such as by email or when accessing web pages. They understand when using various applications, such as web browsers and search engines, where the information is stored and how it is transmitted on request.

What this learning may look like:

Process

Learners are now able to read and reason about more complex code in a visual programming language, including how parallel processes interact. They can practise these skills by completing pencil-and-paper exercises recording step-by-step operation of the code and changes in variables. Alternatively they can verbalise their understanding of the operation of programs in discussion with their peers and teacher, which deepens learning significantly.

The Royal Society of Edinburgh [Starting From Scratch] resources include activities where learners can make predictions about what blocks of Scratch code will do when the flag is clicked. This resource pack also has useful activities for teaching computing technology.

Information

A range of unplugged-style activities can be used to help pupils understand how different sorts of information are represented by the computer.

There are many unplugged activities available from CS Unplugged, Teach London Computing, a Little Bit of CS4Fn, Barefoot Computing and Google's Exploring Computational Thinking. These will be useful across all the SALs and levels.

Computing technology

In addition to a greater depth of understanding of how a computer works, pupils will also be aware of the basics of how computer networks operate. Developing this awareness can be set within the wider context of communicating networks from everyday life.

SAL 3 Second Level	Designing, building and testing computing solutions
Outcome:	**Benchmarks:**
I can create, develop and evaluate computing solutions in response to a design challenge. **TCH 2-15a**	● Creates programs in a visual programming language including variables and conditional repetition ● Identifies patterns in problem solving and reuses aspects of previous solutions appropriately, for example, reuse code for a timer, score counter or controlling arrow keys ● Identifies any mismatches between the task description and the programmed solution, and indicates how to fix them

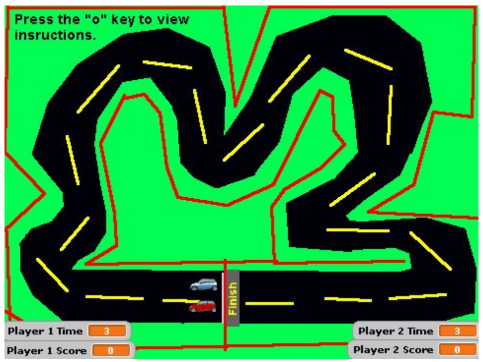
More Information:

Learners use more complex control flow structures. They start to understand how crucial aspects of a process being modelled can be described by variables. Through writing programs which process information, learners begin to understand how program output varies according to program input.

What this learning may look like:

Learners will use their knowledge and understanding from the previous two SALs to design and build computing solutions, such as:

- Games made in Scratch or Kodu to explore choice / decision making processes
- Secret codes created using simple cyphers
- Animations created in a block-based environment like Scratch, Tynker or Hopscotch to explore concepts of repetition and parallel control
- Models built in Minecraft to represent information about the real-world
- Simple web pages made using HTML and CSS to present / represent information (Mozilla Thimble is a great way to make web pages)
- Mobile apps created in App Inventor
- Simple (flat-file) databases



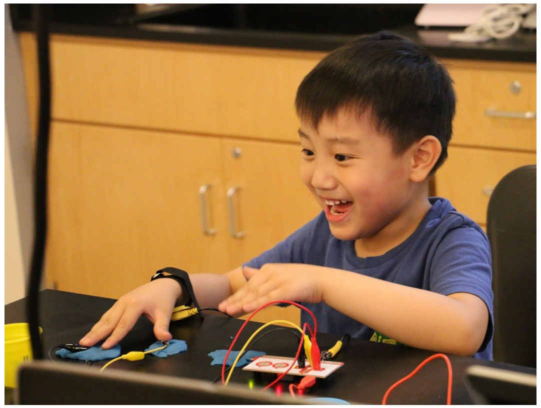
When using a block-based environment, it is best to start with animations using loops and repetition. Later, learners can move onto write algorithms using variables to describe more complex control flow processes including event handling (such as keeping track of health points) and message passing between processes. Creating simple games is a great way to use variables in different ways, such as keeping track of scores.

Learners could demonstrate that they can use different ways to solve and problem and evaluate which way is more effective by creating a racing game in Scratch. For example - would it be a two player race once round a track, a timed single player game or multiple players doing laps? Other considerations are what to do if the car drives off road – will the car slow down or blow up. Pupils will be able to discuss different ways to implement these and to discuss which methods would lead to a more playable game.

Learners can also design and build robotics as computing solutions. Lego Mindstorms is popular, particularly with their annual First Lego League competition. An alternative to Lego is Vex Robotics, and they also run UK and worldwide competitions.

There are also robotic options that require less mechanics, such as Ozobot (where Second level learners could program a maze or racing game with variables using the little robot). Sphero and SPRK robots use block-based apps like Tynker (similar to Scratch). They're also waterproof (and paint-proof!), so learners could build a maze in a paddling pool for the robots to negotiate or draw shapes by driving through a puddle of paint first! These robots should not be just 'driven' but programmed, for example learners could try to develop the fastest or most efficient program that gets a robot through a maze rather than controlling the robot live.

Learners could also program on small computing devices such as Makey Makey kits (which use Scratch to create fun interfaces such as plasticine game controllers and banana pianos!) or BBC Microbit devices (a tiny computer that can respond to shakes or tilting by making noises or showing different patterns of lights)



Resources and Activities

Sources of Support, Resources and Competitions

The following section lists activities and resources that are suitable for teaching the three Significant Aspects of Learning at Early, First and Second Levels. However, there are some websites that we feel are useful for teaching across the levels, or feature a large collection of activities.

Computing At School: http://bit.ly/CSScot45 and **Computing At School Scotland**: http://cas.scot
Computing At School is a grassroots organisation to support the teaching of Computing Science in schools in the UK. Membership is free and open to anyone, including teachers, industry, academics and parents. The CAS Community Forum is a great place to find new resources and ask questions. It's free to join and is a hugely supportive and collaborative forum. CAS also produce Barefoot, Tenderfoot and Quickstart resources for teachers.

Computing Science Unplugged: http://bit.ly/CSScot39
The CS Unplugged resources teach Computer Science through engaging games and puzzles that use cards, string, crayons and lots of running around. The activities introduce students to Computational Thinking concepts without the distraction of having to use computers.

Hello World: http://bit.ly/CSScot46
The Hello World magazine is a computing and digital making magazine for educators that is produced by CAS and the Raspberry Pi Foundation

Hour of Code by **Code.org:** http://bit.ly/CSScot47
Hour of Code takes place each year in December, but the hour-long computer-based and 'unplugged' activities can be used all year round. They are useful for introducing concepts such as selection and repetition in a step-by-step manner before later allowing learners to explore the concepts in a more open environment like Scratch.

Teach London Computing - http://bit.ly/CSScot40
Teaching London Computing is a partnership between Queen Mary University of London and King's College London. The unplugged Computing Science activities do not require a computer and are suitable for Primary pupils. They involve fun activities, puzzles and a bit of magic!

Bebras Computing Competition: http://bit.ly/CSScot48
The UK Bebras Computational Thinking Challenge is a competition with fun logic and problem solving puzzles. It is open to pupils and is staged so that all pupils from P2 to S6 can enter. There are sample questions and past contests on the website which might be of interest too. Your pupils can take the challenge in any 45 minute period during the second and third weeks of November each year.

UK Schools Computer Animation Competition: http://bit.ly/CSScot49
An annual computer animation competition for UK Primary and Secondary pupils. The competition deadline is usually the end of March each year.

Activities for teaching Early Level SALs

Early Level		
Understanding the world through computational thinking	**Understanding and analysing computing technology**	**Designing, building and testing computing solutions**
I can explore computational thinking processes involved in a variety of everyday tasks and can identify patterns in objects or information. **TCH 0-13a**	I understand that sequences of instructions are used to control computing technology. **TCH 0-14a** I can experiment with and identify uses of a range of computing technology in the world around me. **TCH 0-14b**	I can develop a sequence of instructions and run them using programmable devices or equivalent. **TCH 0-15a**
● Identifies and sequences the main steps in an everyday task to create instructions / an algorithm, for example, washing hands ● Classifies objects and groups them into simple categories (links to MNU 0-20a, MNU 0-20b, MNU 0-20c), for example, groups toy bricks according to colour ● Identifies patterns, similarities and differences in objects or information such as colour, size and temperature and simple relationships between them (links to MNU 0-13a)	● Demonstrates an understanding of how symbols can represent process and information ● Predicts what a device or person will do when presented with a sequence of instructions for example, arrows drawn on paper ● Identifies computing devices in the world (including those hidden in appliances and objects such as automatic doors)	● Designs a simple sequence of instructions/algorithm for programmable device to carry out a task for example, directional instructions: forwards/backwards ● Identifies and corrects errors in a set of instructions

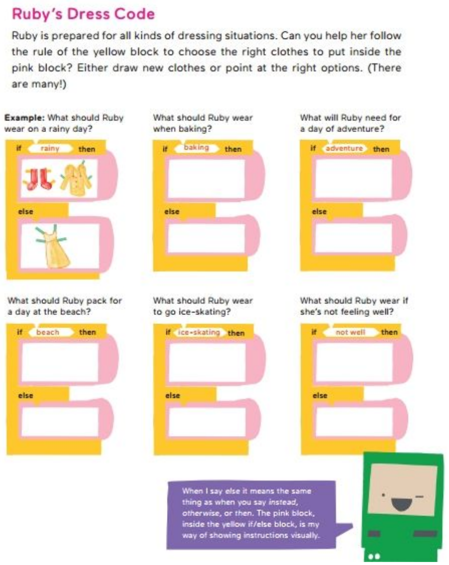
SAL 1 Early Level	Understanding the world through computational thinking		
Resource description	**Source**	**Topic**	**Comment**
This is an unplugged activity in which pupils work in pairs to spot patterns in sets of pictures of objects and think of general statements to describe these things e.g. elephants, cats, cars.	Barefoot Computing - Patterns Unplugged (http://bit.ly/CSScot50)	**Information: Identifying Patterns** By identifying patterns we can make predictions, create rules and solve more general problems. It is a building block of *abstraction*, a key skill in computational thinking.	The emphasis of this activity is on pupils thinking what is the same, what is different and are there general statements they can make about things.
This idea, from the NRich maths project, suggests resources in a nursery/P1 Classroom to encourage open ended maths learning. The computational thinking aspects relate to categorising and sorting.	Socks (http://bit.ly/CSScot5)	**Information - Classifying objects** **Process - sorting** In the early stages, learners can categorise everyday objects such as socks by colour or size. Hanging socks on a washing line in order of size introduces the concepts of ordering and sorting which become important later.	The prompts "How can we remember which sock goes where? (when sorting or ordering)" and "Can we sort them in a different way?" are particularly useful for computational thinking.
This simple sorting activity comes from the Nrich maths project. The computational thinking emphasis is on categorisation of objects.	Packing (http://bit.ly/CSScot52) See also Baskets (http://bit.ly/CSScot53)	**Information - Classifying objects** The ability to categorise objects by their attributes - such as shape, colour, size- is a foundation for handling more complicated information in later stages. It is useful for the children to realise that two objects can be the same on one attribute but different on another.	Try including objects which have similarities on different attributes to start a conversation with the learner about more sophisticated categorisation. For example if you include a red toy van, a red toy ball, a yellow toy ball and green toy car then you can discuss whether which belong together and why.
Another categorising and sorting activity from NRich, which could be combined with health and wellbeing and natural science topics.	Collecting (http://bit.ly/CSScot54) See also Tidying (http://bit.ly/CSScot55)	**Information - Classifying objects** **Process - sorting** Challenging the learners to consider alternative ways of sorting the collection introduces awareness of a powerful computational thinking concept which is relevant to real world data sets. It is also helpful for the learners to hypothetically consider which other objects would belong in a collection.	Useful computational thinking prompts from the activity are "Can you find another way to sort your collection? What if you sorted them into the divided tray? Is that tray big enough? (Can you find one that is?)" and "Is there something else you can think of that could belong here?"

SAL 2 Early Level	Understanding and analysing computing technology		
Resource description	**Source**	**Topic**	**Comment**
This activity involves pupils tinkering with simple robots such as Bee-Bots to find out what they do and how to program them.	[Barefoot Computing - Bee-bots Tinkering](http://bit.ly/CSScot56) (http://bit.ly/CSScot56)	**Process: Control commands** Learners will start to understand the very simple programming language – the command buttons – used by robots like Bee-Bots. Learners will be able to give a robot instructions and predict the effect they have based on the robot's starting state.	The command and challenge word cards could be used for follow up activities, with learners creating instructions on how to get to a secret place in the school or a guide for visitors on how to get from the office to particular places in the building (like your classroom).
Challenge the pupils to predict what Ozobot will do when it is given a picture with coloured codes you have prepared in advance.	[Ozobot - basic training](http://bit.ly/CSScot57) (http://bit.ly/CSScot57)	**Process: simple commands are presented to the robot using colours** Learners will start to understand the simple colour-based programming language for Ozobot robots	Read the basic training information in this resource and identify some simple coloured codes for the children to try out. After they have played for a while, start asking them to predict what the robot will do based on just the coloured codes. This would also be suitable for Level 1.

SAL 3 Early Level	Designing, building and testing computing solutions		
Resource description	**Source**	**Topic**	**Comment**
This activity involves pupils tinkering with simple robots such as Bee-Bots to find out what they do and how to program them.	Barefoot Computing - Bee-bots Tinkering (http://bit.ly/CSScot56)	**Process: Control commands** Learners will start to understand the very simple programming language – the command buttons – used by robots like Bee-Bots. Learners will be able to give a robot instructions and predict the effect they have based on the robot's starting state.	The command and challenge word cards could be used for follow up activities, with learners creating instructions on how to get to a secret place in the school or a guide for visitors on how to get from the office to particular places in the building (like your classroom).
Pupils create sequences of instructions (an algorithm) to draw the shape of a numeral e.g. 3 using a robot such as a beebot.	Barefoot Computing - Bee-bots 1,2,3 (http://bit.ly/CSScot58)	**Process: Algorithms and Programming** An algorithm is a sequence of instructions, or a set of rules, for performing a specific task. Programming in this activity involves taking the algorithm and using it to program a Bee-Bot to navigate a route tracing out the shape of the numeral.	
In this activity pupils design and solve challenges using a programmable toy. To meet the challenges they create sequences of instructions (an algorithm) to navigate a route.	Barefoot Computing - Bee-bots Basics (http://bit.ly/CSScot59)	**Process: Algorithms and Programming** Pupils will write and debug sequences of instructions	

Activities for teaching First Level SALs

First Level		
Understanding the world through computational thinking	**Understanding and analysing computing technology**	**Designing, building and testing computing solutions**
I can explore and comment on processes in the world around me making use of core computational thinking concepts and can organise information in a logical way	I understand the instructions of a visual programming language and can predict the outcome of a program written using the language I can understand how computers process information	I can demonstrate a range of basic problem solving skills by building simple programs to carry out a given task, using an appropriate language
● Follows sequences of instructions/algorithms from everyday situations, for example, recipes or directions, including those with selection and repetition ● Identifies steps in a process and describes precisely the effect of each step ● Makes decisions based on logical thinking including IF, AND, OR and NOT, for example, collecting balls in the gym hall but NOT basketballs, line up if you are left-handed OR have green eyes ● Collects, groups and orders information in a logical, organised way using my own and others' criteria (MNU 1-20a and b)	● Demonstrates an understanding of the meaning of individual instructions when using a visual programming language (including sequences, fixed repetition and selection) ● Explains and predicts what a program in a visual programming language will do when it runs for example, what audio, visual or movement effect will result ● Demonstrates an understanding that computers take information as input, process and store that information, and output the results.	● Simplifies problems by breaking them down into smaller more manageable parts ● Constructs a sequence of instructions to solve a task, explaining the expected output from each step and how each contributes towards solving the task ● Creates programs to carry out activities (using selection and fixed repetition) in a visual programming language ● Identifies when a program does not do what was intended and can correct errors/bugs ● Evaluates solutions/programs and suggests improvements

SAL 1 First Level	Understanding the world through computational thinking		
Resource description	**Source**	**Topic**	**Comment**
The children systematically enumerate the combinations which result from throwing 2 dice.	NRich: Two Dice (http://bit.ly/CSScot60) See also NRich: Button Up (http://bit.ly/CSScot61) and NRich: Beads and Bags (http://bit.ly/CSScot62)	**Process:** Learners can understand and correctly carry out a role assigned to them in a process	This was designed as a maths activity, but the main computational thinking element is systematically trying out and recording the combinations on the dice. This prompt is helpful: How will you know when you've found all the totals?
This is an unplugged activity where learners help Ruby get dressed for different conditions (weather and days of the week). Ruby has a special dress code for each day. IF it's raining THEN wear a rain jacket ELSE ('otherwise') wear a dress	Hello Ruby - Ruby's Dress Code (http://bit.ly/CSScot63) This activity is from the Hello Ruby book (p97) by Linda Liukas. 	**Information: boolean logic and Selection** Conditional statements help computers make decisions and select a set of instructions. This activity uses IF-THEN-ELSE statements to make decisions about what to wear depending on different situations.	Alternatives or extensions to this activity: • Create rules about what food Ruby will eat each day, for example on "Mondays Ruby only eats triangular food", and ask the learners to create a menu for the week • Create rules about which toys Ruby will play with on each day, eg "On Tuesdays Ruby only plays with bumpy toys" and ask the learners to suggest toys for her to play with depending on the day of the week. More opportunities in the classroom for this concept would be asking pupils to line up if they have red hair OR brown eyes, for example.

This is a web-based animated puzzle of a classic logic puzzle. There is a person with a chicken, a fox and some grain trying to get them all across the river in a boat that can't carry everything.	Chicken Fox and Grain puzzle (http://bit.ly/CSScot64)	**Information: Boolean logic and Conditional statements** The learner is given conditional statements about what will happen depending on different combinations of two items being left alone, for example IF chicken and fox are alone THEN the fox will eat the chicken.	
This is an unplugged activity in which pupils work in pairs to spot patterns in sets of pictures of objects and think of general statements to describe these things e.g. elephants, cats, cars.	Barefoot Computing - Patterns Unplugged (http://bit.ly/CSScot50)	**Information: Identifying patterns** By identifying patterns we can make predictions, create rules and solve more general problems.	The emphasis of this activity is on pupils thinking what is the same, what is different and are there general statements they can make about things.
This is an unplugged activity in which pupils create hand clapping, hand tutting or hand jive sequences of movements. Pupils break the sequence of actions down into parts and in so doing are decomposing.	Barefoot Computing - Decomposition Unplugged (http://bit.ly/CSScot65)	**Process: Decomposition and repetition** Pupils link this idea of breaking down a sequence of actions to breaking problems down when creating computer programs such as animations or games.	This activity asks the teacher to compose a dance sequence for the learners to then decompose. It might be easier to use an existing dance instead (eg Macarena, Locomotion, or Bob the Builder's Big Fish Little Fish etc depending on the age of learners). Draw attention to dance moves which are repeated in the same sequence The activity also asks learners to draw the 'commands' but writing a description might be easier for some learners. Discuss with learners the difficulties of coming up with a shared language to communicate the meaning of their 'commands'. Teachers could then ask learners to compose their own sequence using hand tuts.

By teaching this short unplugged activity your pupils will create a set of instructions on how to draw a crazy character and so start to understand what algorithms are.	Barefoot Computing - Crazy Character Algorithms (http://bit.ly/CSScot66)	**Process: Algorithms** An algorithm is a precisely defined sequence of instruction or a set of rules for performing a specific task.	It would be good to follow up this activity by a discussion on why size, scale, angles and precise language are important for giving instructions. This activity could be followed by the CS Unplugged Programming Languages' Marching Orders activity.
In this activity pupils will follow an algorithm to draw pictures constructed from 2D shapes.	Barefoot Computing - 2D Shape Drawing Debugging (http://bit.ly/CSScot67)	**Process: Algorithms** The algorithms learners follow will include errors and they will use logical reasoning to detect and correct these.	
Unplugged activity	CS Unplugged - Create-A-Face (http://bit.ly/CSScot68)	Explore algorithms by making an robot face out of card, tubes and students. Program it to react to different kinds of sounds (nasty, nice or sudden) and show different emotions (sad, happy, surprised).	Then think up some other facial expressions and program rules to make the face respond to sounds with the new expressions. This links to the Emotional Machine activity in SAL 2.
In this unplugged activity pupils spot patterns in pairs of similar recipes to identify common steps that they can reuse in new recipes that they create. Example sets of simple recipes are provided on how to make sandwiches, pizza and milkshakes.	Barefoot Computing - Patterns unplugged: Recipes (http://bit.ly/CSScot69)	**Process: Identifying Patterns** The emphasis of this activity is on pupils thinking what is the same, what is different and are there general common elements that they can reuse.	This activity would suit learners working at either Level 1 or 2.
Story	*Computational Fairy Tales* book: The Marvelous IF-ELSE Life of the King's Turtle (http://bit.ly/CSScot70)	**Process: Conditions**	You could ask the children to write IF-THEN-ELSE rules to describe the behaviour of their pets at home
Story	*Computational Fairy Tales* book: Learning IF-ELSE the Hard Way (http://bit.ly/CSScot71)	**Process: Conditions**	You could ask the children to write IF THEN ELSE rules about classroom behaviour. Encourage them to chain on additional "IFs" to capture more complex rules

Story	Computational Fairy Tales book: While Loops and Dizziness (http://bit.ly/CSScot72)	**Process: Loops**	Introduction to a simple while loop. Suitable quick example for younger learners
Story	*Computational Fairy Tales* book:Loops and Making Horseshoes (http://bit.ly/CSScot73)	**Process: Loops**	This story draws attention to the difference between "for" loops and "while" loops. You could ask the children to make a Scratch program in which a blacksmith character hits metal with a hammer 10 times (SAL 3).
This NRich activity extends the concept of a sorted collection of objects by considering a queue.	NRich (http://bit.ly/CSScot74a): Queueing (http://bit.ly/CSScot74b)	**Information - queue** This activity draws attention to the rules of queues such as which item in the queue is processed first. Queues are frequently used to decide the order in which data processed in later stages of computational thinking, and are familiar from everyday life.	These prompts are particularly useful "Is the first in the queue always at the front? Is the last in the queue always at the back?". The queue has a first in, first out rule. Extension: Not all approaches to processing collections of information have this rule though. Ask the children to compare it to a stack of plates: here the rule is usually last in, first out. Unless you want a lot of broken plates.
Story	Computational Fairy Tales book: Stacks, Queues, Priority Queues, and the Prince's Complaint Line (http://bit.ly/CSScot75)	**Information: queue**	This story follows up on the queues concept from the above NRich activity by comparing what happens when you use different strategies for dealing with queues or stacks of people who all want something. You could try using these strategies when children want attention in the class!

SAL 2 First Level	Understanding and analysing computing technology		
Resource description	**Source**	**Topic**	**Comment**
Book	Lift-the-flap Computers and Coding (http://bit.ly/CSScot76) "What's Inside" page	Learners understand the main features of a computer, including input, processing and output, and can identify these in a range of digital technologies.	This goes into a surprising amount of detail - suitable for the upper end of Level 1 and into Level 2.
An interactive presentation labelling the parts of a computer plus a game.	BBC Bitesize - main parts of a computer (http://bit.ly/CSScot77)	Learners understand the main features of a computer, including input, processing and output, and can identify these in a range of digital technologies.	You could also have a display with old computer parts and encourage the children to handle them.
This iPad/Android tablet programming activity involves pupils tinkering with ScratchJr to find out what it does and how to create programs in it.	Barefoot Computing - ScratchJr Tinkering (http://bit.ly/CSScot78)	**Process: Programming** Pupils will gain familiarity with the ScratchJr environment and commands	Think of challenges for learners, such as "Can you make the cat spin around / make a noise / do something unexpected". Ask learners to share their work with others and talk about the commands they used. This would help to focus the conversation on the code constructs rather than the effects of the code.
This computer-based programming activity involves your pupils tinkering with Kodu to find out what it does and how to create programs in it.	Barefoot Computing - Kodu Tinkering (http://bit.ly/CSScot79)	**Process: Programming** Pupils will gain familiarity with the Kodu environment and commands	The Microsoft Kodu site would be a good follow on from this activity.
Unplugged activity	CSUnplugged - Binary numbers (http://bit.ly/CSScot80)	**Information: Binary numbers and data representation** "The binary number system plays a central role in how information of all kinds is stored on computers. Understanding binary can lift a lot of the mystery from computers, because at a fundamental level they're really just machines for flipping binary digits on and off."	This unplugged activity looks at how numbers are represented in a computer "There are several activities on binary numbers in this document, all simple enough that they can be used to teach the binary system to anyone who can count!"

Reading diagrams	NRich: Carroll diagrams (http://bit.ly/CSScot81)	**Information:** Learners can understand diagrams which illustrate key aspects of information	This is very much linked to maths learning and the logical thinking exercises from SAL 1.
Introduction to writing algorithms	Tour Guide (http://bit.ly/CSScot82)	Creating a simple algorithm to help tourists get from their hotel to all the city sights and back to their hotel.	This activity explores the benefits of creating algorithms and thinks about the benefits of efficiency (with shortest algorithm and shortest path)
Unplugged activity to program a cardboard robot	Emotional Machine (http://bit.ly/CSScot83)	This is a very simple way to introduce the idea of programs and sequences of instructions. The class program a card robot face to show different emotions one after another.	This activity follows on from the Create A Face (in SAL 1, first level)

SAL 3 First Level	Designing, building and testing computing solutions		
Resource description	**Source**	**Topic**	**Comment**
This iPad/Android tablet programming activity involves your pupils tinkering with ScratchJr to find out what it does and how to create programs in it.	Barefoot Computing - ScratchJr Tinkering (http://bit.ly/CSScot78)	**Process: Programming** Pupils will gain familiarity with the ScratchJr environment and commands	Think of challenges for learners, such as "Can you make the cat spin around / make a noise / do something unexpected". Ask learners to share their work with others and talk about the commands they used. This would help to focus the conversation on the code constructs rather than the effects of the code.
In this iPad/Android tablet programming activity pupils, in pairs, create a simple animation program of a knock knock joke. They use a storyboard to create their design, write the code in ScratchJr, debug and evaluate.	Barefoot Computing - ScratchJr Jokes (http://bit.ly/CSScot84)	**Process: Programming** Pupils will have to control the timing and order of the two sprites saying the knock, knock joke lines. In this ScratchJr activity the 'wait' command is used to sequence the events.	

Activities for teaching Second Level SALs

Second Level		
Understanding the world through computational thinking	**Understanding and analysing computing technology**	**Designing, building and testing computing solutions**
I understand the operation of a process and its outcome. I can structure related items of information.	I can explain core programming language concepts in appropriate technical language I understand how information is stored and how key components of computing technology connect and interact through networks	I can create, develop and evaluate computing solutions in response to a design challenge
● Compares activities consisting of a single sequence of steps with those consisting of multiple parallel steps, for example, making tomato sauce and cooking pasta to be served at the same time ● Identifies algorithms / instructions that include repeated groups of instructions a fixed number of times and/or loops until a condition is met ● Identifies when a process is not predictable because it has a random element, for example, a board game which uses dice ● Structures related items of information, for example, a family tree (MNU 2-20b) ● Uses a recognised set of instructions / an algorithm to sort real worlds objects, for example, books in a library or trading cards	● Explains the meaning of individual instructions (including variables and conditional repetition) in a visual programming language ● Predicts what a complete program in a visual programming language will do when it runs, including how the properties of objects for example, position, direction and appearance, change as the program runs through each instruction ● Explains and predicts how parallel activities interact ● Demonstrates an understanding that all computer data is represented in binary, for example, numbers, text, black and white graphics. ● Describes the purpose of the processor, memory and storage and the relationship between them ● Demonstrates an understanding of how networks are connected and used to communicate and share information, for example, the internet	● Creates programs in a visual programming language including variables and conditional repetition ● Identifies patterns in problem solving and reuses aspects of previous solutions appropriately, for example, reuse code for a timer, score counter or controlling arrow keys ● Identifies any mismatches between the task description and the programmed solution, and indicates how to fix them

SAL 1 Second Level	Understanding the world through computational thinking		
Resource description	**Source**	**Topic**	**Comment**
In this activity pupils explain the rule for a number sequence and predict which number(s) comes next.	Barefoot Computing- Logical number sequences (http://bit.ly/CSScot85)	**Process: Algorithms. Logic, Patterns** Learners extend their knowledge of simple rule based algorithms. They also use logical reasoning as they work out and explain their algorithms.	
This is an unplugged activity in which pupils work in pairs to complete sudoku puzzles. The emphasis of this activity is on pupils using logical reasoning to solve the puzzles – pupils have to explain to their partner how they have worked out each number they add to the sudoku grid.	Barefoot Computing - Logical Reasoning unplugged (http://bit.ly/CSScot86)	**Process: Logical Reasoning** In this activity pupils use logical reasoning as they analyse the sudoku squares to work out which number to add next. Pupils are encouraged to regularly explain their thinking, to help develop both their logical reasoning and their ability to articulate such reasoning.	It may help learners to discuss the logical rules (or 'heuristics') for solving sudoku puzzles beforehand. Learners could also discuss strategies where the number in a square is not uniquely determined. Learners that need an additional challenge could work on 9x9 grid sudoku puzzles. This activity could be followed up by using Logic Grid puzzles in class.
In this unplugged activity pupils spot patterns in pairs of similar recipes to identify common steps that they can reuse in new recipes that they create. Example sets of simple recipes are provided on how to make sandwiches, pizza and milkshakes.	Barefoot Computing - Patterns unplugged: Recipes (http://bit.ly/CSScot69)	**Process: Identifying Patterns** The emphasis of this activity is on pupils thinking what is the same, what is different and are there general common elements that they can reuse.	This activity would suit learners working at either Level 1 or 2.
This is an unplugged activity in which pupils learn about variables by keeping score for a game.	Barefoot Computing - Variables unplugged (http://bit.ly/CSScot87)	**Information: Variables** Pupils learn why variables are needed, how they are created, how they store data, and how this data may be used by a computer program as it runs.	

Unplugged drawing activity. In pairs students give instructions to draw a shape to their partner.	CS Unplugged - Programming Languages (http://bit.ly/CSScot88)	**Programming Languages - Drawing** Computer programs are sequences of instructions that the computer must follow. This Marching Orders activity demonstrates some of the issues that arise when we try to give precise instructions to achieve a desired outcome.	Variations to this activity are given but students can also work on describing more challenging images such as doodle monsters (http://bit.ly/CSScot89).
Unplugged activity. Students give instructions (to a teacher-bot!) on how to make a jam sandwich using only a fixed set of allowed words.	Code-IT Jam Sandwich (http://bit.ly/CSScot90)	**Programming Languages - Jam Sandwich** This activity demonstrates some of the issues that arise when we try to give precise instructions to achieve a desired outcome.	This activity is best done by the whole class together giving instructions to the teacher (and ideally it all goes disastrously wrong due to inaccuracy!). Students can then perfect their algorithm and try to optimise it.
Story	Computational Fairy Tales: Goldilocks and the Two Boolean Bears (http://bit.ly/CSScot91)	**Information:Boolean logic**	A twist on the class Goldilocks story in which two bears have exactly opposite preferences. The pupils could draw a truth table for the bears' preferences.
Story	Computational Fairy Tales: The Town of Bool (http://bit.ly/CSScot92), The Gates of XOR (http://bit.ly/CSScot93), The Valley of NAND and NOR (http://bit.ly/CSScot94)	**Information:Boolean logic**	The Town of Bool illustrates how real life doesn't tend to work with binary logic - it is really tedious. The Gates of XOR and the Valley of NAND and NOR illustrate more advanced logical operators which might appeal to learners who have found the logic material easy to grasp so far. Or to kids who love to trip other people up with pedantic logic.
Story	Computational Fairy Tales: The Tortoise, the Hare, and 50000 Ants (http://bit.ly/CSScot95)	**Process: Algorithms/ parallel processes**	It would be a good idea to tell the story of the Tortoise and the Hare first. This story illustrates why doing tasks in parallel is a good idea - it saves time and in some cases makes it possible to solve a problem in the first place. None of the ants would be able to run 5000m by himself, but they can do the same distance as a team. This story develops an understanding of how solving the same problem using different approaches will result in more or less *efficient* solutions.

Story	Computational Fairy Tales book: Bullies, Bubble Sort, and Soccer Tickets (http://bit.ly/CSScot96)	**Process: Sorting (bubble sort)**	If you're not using the Fairy Tales book as a running example, you could adapt this story slightly into an unplugged exercise.
Story	Computational Fairy Tales book: Bog dragons don't support multi-threading (http://bit.ly/CSScot97)	**Process: Multi-tasking/ parallel processing**	A computer with a single processor can only do one thing at a time. This means it needs to rapidly switch between tasks to enable the user to read email, browse the web and have a document open all at the same time. A system which can handle this supports *multi-threading*. A system which can't is pretty useless, like the bog dragon in this story. There are various traditional folk tales in which the monster can be easily distracted to let the hero get away.
Unplugged activity	CS Unplugged - Searching Algorithms (http://bit.ly/CSScot98)	**Process: Searching Algorithms with Battleships** "Searching for a keyword or value is the basis of many computing applications, whether on an internet search engine or looking up a bank account balance."	"This activity explores the main algorithms that are used as the basis for searching on computers, using different variations on the game of battleships."
Unplugged activity	CS Unplugged - Sorting Algorithms (http://bit.ly/CSScot99)	**Process: Sorting Algorithms** "Almost any list that comes out of a computer is sorted into some sort of order, and there are many more sorted lists inside computers that the user doesn't see. Many clever algorithms have been devised for putting values into order efficiently."	"In this activity students compare different algorithms to sort weights in order."
Unplugged activity	CS Unplugged - Sorting Networks (http://bit.ly/CSScot100)	**Process: Parallel Sorting** "To make computers go faster, it can be a lot more effective to have several slower computers working on a problem than a single fast one. This raises questions about how much of the computation can be done at the same time."	"Here we use a fun team activity to demonstrate an approach to parallel sorting. It can be done on paper, but we like to get students to do it on a large scale, running from node to node in the network."

Puzzles	Logic grid puzzles (http://bit.ly/CSScot101)	**Information: boolean logic**	Logic grid puzzles are a nice way of exploring use of logic and deduction. Clues are given that give partial information and the learner needs to work out the rest of the information using logical thinking.
Unplugged activity	CS Unplugged - Information Theory (http://bit.ly/CSScot102)	**Information: Information Theory** "Computers are all about storing and moving information, but what actually is information? How do we measure the amount of information in a message?"	"This activity uses some intriguing variations on the game of 20 questions to demonstrate how we can quantify information content, which in turn shows us how to store and share it efficiently."
Unplugged activity	CS Unplugged - Finite State Automata (http://bit.ly/CSScot103)	**Process: Finite State Automata** "Finite-state automata are used in computer science to help a computer process a sequence of characters or events. Finite state automata (FSAs) sound complicated, but the basic idea is as simple as drawing a map."	"This activity is based around a fictitious pirate story which leads to the unlikely topic of reasoning about patterns in sequences of characters"
Unplugged activities in which learners reason about chance while playing common games	NRich: A bit of a dicey problem (http://bit.ly/CSScot104) NRich: Rock, paper, scissors (http://bit.ly/CSScot105)	**Process: predictable behaviour**	Randomness in a process makes it unpredictable in the sense that a random process may produce varied output when given the same input. This activity develops the learner's' understanding of the range of outputs to expect from the simple random process of throwing a die. This is a useful prompt "Do you have more chance of getting one answer than any other?"
Groups of learners work together to design a board game. The emphasis should be on the rules of the game, whether it involves chance, and whether and how it ends.	Design a board game (http://bit.ly/CSScot106)	**Process: predictable behaviour and when processes end**	Sessions 1-2 in this plan are most relevant for computational thinking. Describe the rules of your game [this is good practice for clearly stating an algorithm] What causes the game to finish? Can you be sure the game will finish? How can you change your game to make sure it finishes? [Hint: adding a timer is an easy way! This activity would also be suitable for SAL 3.

SAL 2 Second Level	Understanding and analysing computing technology		
Resource description	**Source**	**Topic**	**Comment**
Book	Lift-the-flap Computers and Coding "How Computers Think" page and "Computer Language" page (http://bit.ly/CSScot107)	Learners have the **knowledge and understanding** of how more complex digital technologies work. Digital information representations	This resource page covers how information is represented and stored by the computer
Book	Lift-the-flap Computers and Coding "Using the internet" (http://bit.ly/CSScot107)	Learners have **knowledge and understanding** of computer networks.	This page explains what happens when you open a webpage and the basics of how the internet works
This computer-based programming activity involves your pupils tinkering with Scratch to find out what it does and how to create programs in it.	Barefoot Computing - Scratch Tinkering (http://bit.ly/CSScot108)	**Process: Programming** Pupils will gain familiarity with the Scratch environment and commands	Additional questions / prompts for this activity are "What do these commands do?" and "What would happen if we changed the order of the blocks/commands?" Learners can showing their creations to other learners. Get everyone to display their programs full screen and ask learners to walk around and predict which commands were used to give the effects they see in the program. Alternatively they could look at the code without pressing the green flag and predict what will happen when the program runs. Stretch and Challenge tasks, or next steps for the whole class, can be sourced from the excellent Creative Computing guide or the Scratch cards.
In this computer-based programming activity, pupils design algorithms to draw patterns made of simple shapes before writing a Scratch program to draw their shapes.	Barefoot Computing - Shapes and Flowers Repetition (http://bit.ly/CSScot109)	**Process: Programming - Repetition** Pupils will learn about using repeat commands in Scratch	Stretch and Challenge tasks, or follow on activities for the whole class, can be sourced from the excellent Creative Computing guide or the Scratch cards.

In this computer-based programming activity pupils are challenged to detect and correct the error in a number of water cycle Scratch programs (debugging).	Barefoot Computing - Bug in the Water Cycle (http://bit.ly/CSScot110)	**Process: Programming - debugging** Pupils use logical reasoning to correct errors (debug) in Scratch programs, comparing what the program should do with what it does do, and systematically homing in on the error (bug) by 'thinking through' the code in the program.	
This is computer-based activity introduces pupils to HTML. Pupils learn that web pages are written using HTML and become familiar with basic HTML tags by remixing web pages using Mozilla X-Ray Goggles.	Barefoot Computing - Introduction to HTML (http://bit.ly/CSScot111)	**Process: Programming - HTML** Web pages are written using a special language called HTML. HTML tells the web browser how to structure and display the page. HTML stands for HyperText Mark-up language.	
In this unplugged activity pupils are assigned roles as different digital devices in a human model of the internet and learn how the internet provides access to the WWW (an internet service) as they pass data between them.	Barefoot Computing - Modelling the Internet (http://bit.ly/CSScot112)	**Networking** The internet is a vast network of computers and other devices connected across the world. Pupils explore the difference between the internet and the world wide web (WWW).	
In this activity pupils go on a hunt around their school to discover, and map the location of, devices connected to their school's network. Pupils then learn about the role of each device by either conducting web-based research or using the matching activity included.	Barefoot Computing - Network Hunt (http://bit.ly/CSScot113)	**Networking** A computer network is a collection of computer systems (http://bit.ly/CSScot148) and other devices connected together to 'talk' to each other by exchanging data.	Pupils will discover that a network is typically made up of a server (a computer which provides services to a network) and clients (computers which use the services on the network). Pupils also learn about other devices on a local area network, such as a switch and wifi points which enable computers to communicate by exchanging data.

In this computer-based activity, pupils learn about the basics of how search engines use web crawlers to index the world wide web (WWW) and how this is used to select search results. Pupils act like web crawlers themselves, indexing a very small portion of the WWW, and they then use this index to respond to search queries.	Barefoot Computing - Selecting Search (http://bit.ly/CSScot114)	**Networking - Search engines** Search engines are programs designed to help users find information on the world wide web. They do this by building up an index of the web using web crawlers. Web crawlers are programs which move across the web by following the links between pages. They take copies of the web pages they visit to build up a search engine's index.	
This is an unplugged activity in which pupils learn about some of the main factors which influence how a search engine ranks a web page. Pupils create paper-based 'web pages' in groups on a current topic they are studying. They then discover how their web pages would rank when searching for keywords relating to their content.	Barefoot Computing - Ranking Search (http://bit.ly/CSScot115)	**Networking - Search engines** To rank the search results the search engine program analyses the content of each web page to determine how relevant they are to your search.	
This computer-based programming activity is an investigation of different input devices, where pupils are challenged to create a Scratch program that uses the input from a device in a short piece of code.	Barefoot Computing - Investigating Inputs (http://bit.ly/CSScot116)	**Process: Programming and Input devices** An input device is a digital device that takes data from the outside world and converts it into a format that a computer system can use, such as a keyboard, microphone or mouse.	If your school doesn't have technology such as picoboards or WeDo kits, don't forget that a mouse is an input device too! The Scratch commands for following the mouse are quite fun too!
In this computer-based programming activity pupils learn about output devices and create a program to control a LEGO Education WeDo motor using Scratch.	Barefoot Computing - Investigating Outputs (http://bit.ly/CSScot117)	**Process: Programming output devices** An output device is a digital device that takes data from a computer system and converts it for use in the outside world, such as sound (with speakers or headphones), vision (using a computer monitor) or motion (using a motor)	This activity is not possible without access to a WeDo kit. If you want to explore output devices and motors then Ohbot is an alternative to WeDo. Speakers are also output devices and it is quite fun to explore recording and using sound effects in Scratch (using a microphone as an input device to record speech or noises)

In this computer-based programming activity pupils create a sound monitor for their classroom.	Barefoot Computing - Classroom Sound Monitor (http://bit.ly/CSScot118)	**Programming - control using sensors** The sound monitors they create are examples of control (http://bit.ly/CSScot140) programs – they take information from an input sensor (http://bit.ly/CSScot141) (a microphone), and use this information to alter the output (http://bit.ly/CSScot142) of the program (displaying a warning message if pupils are too noisy).	
Story	Computational Fairy Tales: Using Binary to Warn of Snow Beasts (http://bit.ly/CSScot119)	**Information: Binary representations**	A story which illustrates how information can be encoded in a binary representation to be more space efficient
Unplugged activity	CSUnplugged - Binary numbers (http://bit.ly/CSScot80)	**Information: Binary numbers and data representation** “The binary number system plays a central role in how information of all kinds is stored on computers. Understanding binary can lift a lot of the mystery from computers, because at a fundamental level they’re really just machines for flipping binary digits on and off.”	This unplugged activity looks at how numbers are represented in a computer “There are several activities on binary numbers in this document, all simple enough that they can be used to teach the binary system to anyone who can count!”
An unplugged activity to make loom band bracelets with letters in binary.	Binary Loom Bands (http://bit.ly/CSScot120) YouTube video by Karen Petrie with worksheet link in the video description notes	**Information: Binary numbers and text representation (ASCII)** When we press keys on a keyboard or send a text message, those letters are sent as binary numbers based on a standard code (For example ‘A’ is 65 in binary, ‘z’ is 122)	This activity shows you how to convert your initials into binary then make a loom band bracelet which incorporates them.

Unplugged activity	CS Unplugged - Image Representation (http://bit.ly/CSScot121)	**Information: Image representation** "Images are everywhere on computers. Some are obvious, like photos on web pages and icons on buttons, but others are more subtle: a font is really a collection of images of characters, and a fax machines is really a computer that is good at scanning and printing."	"This activity explores how images are displayed, based on the pixel as a building block. In particular, the great quantity of data in an image means that we need to use compression to be able to store and transmit it efficiently. The compression method used in this activity is based on the one used in fax machines, for black and white images."
Unplugged activity	CS Unplugged - Text compression (http://bit.ly/CSScot122)	**Information: Text compression** "Since computers only have a limited amount of space to hold information, they need to represent information as efficiently as possible. This is called compression. By coding data before it is stored, and decoding it when it is retrieved, the computer can store more data, or send it faster through the Internet."	"Children's rhymes and stories are good examples for text compression, because they often involve repeated words and sequences."
Unplugged activity	CS Unplugged - Error Detection (http://bit.ly/CSScot123)	**Information: Error detection** "When data is stored on a disk or transmitted from one computer to another, we usually assume that it doesn't get changed in the process. But sometimes things go wrong and the data is changed accidentally."	"This activity uses a magic trick to show how to detect when data has been corrupted, and to correct it."
Unplugged game	CS Unplugged - Routing and Deadlock (http://bit.ly/CSScot124)	**Network Routing and Deadlock** "Computer networks are based on passing messages from computer to computer. This sounds simple in principle, but in practice all sorts of contention and bottlenecks can occur."	"Computer networks are based on passing messages from computer to computer. This sounds simple in principle, but in practice all sorts of contention and bottlenecks can occur." "This activity gives some first hand experience of such issues, with a game for a group of students."

Unplugged activity	CS Unplugged - Minimal Spanning Trees (http://bit.ly/CSScot125)	**Minimal Spanning Trees** "Our society is linked by many networks: telephone networks, utility supply networks, computer networks, and road networks. For a particular network there is usually some choice about where the roads, cables, or radio links can be placed. We need to find ways of efficiently linking objects in a network."	"This puzzle shows students the decisions involved in linking a network between houses in a muddy city." It can lead on to a discussion of finding the shortest path for networks, pipes and journeys, and why this is important."
Unplugged activity	CS Unplugged - Network Communication Protocols (http://bit.ly/CSScot126)	**Network Communication Protocols** "Computers talk to each other over the internet via messages. However, the internet is not reliable and sometimes these messages get lost. There are certain bits of information we can add to messages to make sure they are sent. This information makes up a protocol."	"In this Tablets of Stone activity students consider how different methods of communication operate successfully. By looking at rules and procedures in place, students are introduced to communication protocols."

SAL 3 Second Level	Designing, building and testing computing solutions		
Resource description	**Source**	**Topic**	**Comment**
A huge variety of computer-based Scratch programming activities. The authors say "No prior experience with computer programming is required, only a sense of adventure!"	Harvard University - Creative Computing (http://bit.ly/CSScot127)	**Process: Programming** Pupils will gain familiarity with the Scratch environment and commands while learning Computing Science concepts and being creative at the same time.	This is an excellent guide for teachers wanting to teach Computing concepts using Scratch to their learners. There are teacher activity sheets and handouts for learners. There are activities for those who are completely new to Scratch and also some interesting and fun challenges for those who are very experiences with the programming environment.
Unit 4 focuses on making games and introduces data handling including lists	Harvard University - Creative Computing: Unit 4 (http://bit.ly/CSScot127)	**Information:** Through writing programs which process information, learners begin to understand how program output varies according to program input.	Programs which handle data are more challenging, but also closer to solving real world programming problems. This unit would work with more advanced/confident learners towards the end of the level.
A set of introductory activities for learning Computing Science through Scratch.	Royal Society of Edinburgh - Introduction to Computing Science: Starting From Scratch (http://bit.ly/CSScot128)	**Process: Programming** Pupils will gain familiarity with the Scratch environment and commands as well as learning about core CS concepts	This exemplification resource was originally written for CfE Level 3 but would be very suitable for Level 2 SAL 3 within this Progression Framework.
This computer-based programming activity involves your pupils tinkering with Scratch to find out what it does and how to create programs in it.	Barefoot Computing - Scratch Tinkering (http://bit.ly/CSScot108)	**Process: Programming** Pupils will gain familiarity with the Scratch environment and commands	Additional questions / prompts for this activity are "What do these commands do?" and "What would happen if we changed the order of the blocks/commands?" Learners can showing their creations to other learners. Get everyone to display their programs full screen and ask learners to walk around and predict which commands were used to give the effects they see in the program. Alternatively they could look at the code without pressing the green flag and predict what will happen when the program runs. Stretch and Challenge tasks, or next steps for the whole class, can be sourced from the excellent Creative Computing guide (http://bit.ly/CSScot127) or the Scratch cards (http://bit.ly/CSScot129).

This computer-based programming activity involves your pupils tinkering with Kodu to find out what it does and how to create programs in it.	Barefoot Computing - Kodu Tinkering (http://bit.ly/CSScot130)	**Process: Programming** Pupils will gain familiarity with the Kodu environment and commands	The Microsoft Kodu site (http://bit.ly/CSScot131) would be a good follow on from this activity.
In this computer-based programming activity, pupils design algorithms to draw patterns made of simple shapes before writing a Scratch program to draw their shapes.	Barefoot Computing - Shapes and Flowers Repetition (http://bit.ly/CSScot109)	**Process: Programming - Repetition** Pupils will learn about using repeat commands in Scratch	Stretch and Challenge tasks, or follow on activities for the whole class, can be sourced from the excellent Creative Computing guide (http://bit.ly/CSScot127) or the Scratch cards (http://bit.ly/CSScot129).
In this computer-based programming activity pupils program an animation in Scratch illustrating the steps in fossil formation.	Barefoot Computing - Fossil Formation animation (http://bit.ly/CSScot132)	**Process: Programming - sequences and implementing algorithms** Pupils will learn that programming is the process of implementing algorithms as code, and about sequencing commands in Scratch.	Stretch and Challenge tasks, or follow on activities for the whole class, can be sourced from the excellent Creative Computing guide (http://bit.ly/CSScot127) or the Scratch cards (http://bit.ly/CSScot129).
In this computer-based programming activity pupils program an animation in Scratch of a Viking raid.	Barefoot Computing - Viking Raid animation (http://bit.ly/CSScot133)	**Process: Programming - sequences and implementing algorithms** Pupils will learn that programming is the process of implementing algorithms as code, and about sequencing commands in Scratch.	Stretch and Challenge tasks, or follow on activities for the whole class, can be sourced from the excellent Creative Computing guide (http://bit.ly/CSScot127) or the Scratch cards (http://bit.ly/CSScot129).
In these computer-based programming activity pupils program a Maths quiz in Scratch.	Barefoot Computing - Scratch Maths Quiz - selection (http://bit.ly/CSScot134a) Barefoot Computing - Scratch Maths Quiz - variables (http://bit.ly/CSScot134b)	**Process: Programming - selection and variables** Selection (also known as conditions) allows the flow of the program to be altered depending on the player's answers to questions. Pupils then learn to use variables in Scratch to make a scoring system for their maths quiz.	Stretch and Challenge tasks, or follow on activities for the whole class, can be sourced from the excellent Creative Computing guide (http://bit.ly/CSScot127) or the Scratch cards (http://bit.ly/CSScot129).

In this computer-based programming activity pupils create an animation of a poem using Scratch.	Barefoot Computing - Animated Poem (http://bit.ly/CSScot136)	**Process: Programming and Problem Solving - Decomposition** Pupils will learn about decomposition. Decomposition is breaking something down into smaller parts to help solve a problem or undertake a task.	Stretch and Challenge tasks, or follow on activities for the whole class, can be sourced from the excellent Creative Computing guide (http://bit.ly/CSScot127) or the Scratch cards (http://bit.ly/CSScot129).
In this computer-based programming activity pupils create a simulation (http://bit.ly/CSScot139) of the Earth orbiting the Sun using Scratch. Pupils decide what the purpose of the simulation is and who is the intended audience.	Barefoot Computing - Solar System Simulation (http://bit.ly/CSScot137)	**Process: Programming and Problem Solving - Abstraction** Pupils decide what the most important aspects of the simulation are, and in so doing they are abstracting (http://bit.ly/CSScot138).	Stretch and Challenge tasks, or follow on activities for the whole class, can be sourced from the excellent Creative Computing guide (http://bit.ly/CSScot127) or the Scratch cards (http://bit.ly/CSScot129).
This computer-based programming activity is an investigation of different input devices, where pupils are challenged to create a Scratch program that uses the input from a device in a short piece of code.	Barefoot Computing - Investigating Inputs (http://bit.ly/CSScot116)	**Process: Programming and Input devices** An input device is a digital device that takes data from the outside world and converts it into a format that a computer system can use, such as a keyboard, microphone or mouse.	If your school doesn't have technology such as picoboards or WeDo kits, don't forget that a mouse is an input device too! The Scratch commands for following the mouse are quite fun too!
In this computer-based programming activity pupils learn about output devices and create a program to control a LEGO Education WeDo motor using Scratch.	Barefoot Computing - Investigating Outputs (http://bit.ly/CSScot117)	**Process: Programming output devices** An output device is a digital device that takes data from a computer system and converts it for use in the outside world, such as sound (with speakers or headphones), vision (using a computer monitor) or motion (using a motor)	This activity is not possible without access to a WeDo kit. If you want to explore output devices and motors then Ohbot is an alternative to WeDo. Speakers are also output devices and it is quite fun to explore recording and using sound effects in Scratch (using a microphone as an input device to record speech or noises)

In this computer-based programming activity pupils create a sound monitor for their classroom.	Barefoot Computing - Classroom Sound Monitor (http://bit.ly/CSScot118)	**Process: Programming - control using sensors** The sound monitors they create are examples of control (http://bit.ly/CSScot140) programs – they take information from an input sensor (http://bit.ly/CSScot141) (a microphone), and use this information to alter the output (http://bit.ly/CSScot142) of the program (displaying a warning message if pupils are too noisy).	
In this computer-based programming activity pupils create a simple game in Scratch.	Barefoot Computing - Make a Game project (http://bit.ly/CSScot143)	**Process: Programming** Pupils design their game and create artwork for their background and main character. They then write and debug their code, and finally they present and evaluate their games.	Preparation activities can be sourced from the excellent Creative Computing guide (http://bit.ly/CSScot127) or the Scratch cards (http://bit.ly/CSScot129). Both of these sites have useful activities for learning core skills for Scratch game design. It will help learners to develop techniques that can be reused such as control methods (using keys or mouse to move a sprite), game mechanisms (scores, lives, timers etc) and sensing techniques (using 'touching sprite' or 'touching colour' blocks to sense when a sprite is over another sprite or a particular part of the game space). For this activity ask learners to create a very specific game type rather than allowing a wide choice.
In this computer-based programming activity pupils create a simple game in Kodu.	Barefoot Computing - Kodu Game - selection (http://bit.ly/CSScot144)	**Process: Programming - selection** Pupils create a design for their game, which includes rule-based algorithms describing how it will be played and a sketch of the Kodu world it will be played in. Pupils then create the Kodu world, implement their algorithms as code and play and evaluate each other's games.	The Microsoft Kodu site (http://bit.ly/CSScot131) is a good source of support materials and preparation activities before moving on to this activity.

Glossary

Abstraction
Abstraction is a key concept in computer science. Abstraction is about improving understanding by separating core concepts from inessential detail. When solving complex problems, abstraction enables us to focus on the more important aspects, thus helping to manage complexity. For example, a timetable is an abstraction of what a pupil will do in a typical week. It shows the pupils details of the subjects they will learn, who will teach them and when and where the lessons happen. Details such as learning objectives for individual lessons are not included, as this is not important for the intended purpose of the timetable. More information on abstraction (http://bit.ly/CSScot138)

Algorithm
An algorithm is a precisely-defined sequence of instructions which is used to describe a process: a set of rules to describe how to get something done. Algorithms are usually written for a human, rather than for a computer, to understand. Algorithms normally have a start point and an end point, usually have some input, and are expected to finish with a correct outcome. As they share many of these properties with computer programs, they are often easy to translate into a programming language. More information on algorithms (http://bit.ly/CSScot145)

Binary
Most computers use binary numbers to represent information, through using “on” and “off” circuits to represent binary numbers.. We are more used to counting in the decimal number system, because we have 10 fingers to count on! Computers use binary to store and represent not just numbers, but also letters, images, videos, blocks in a visual programming language, and even the instructions that they can execute.

Bugs / Debugging
A bug in a computer program happens when the programmer has made a mistake or has missed out a bit of code, causing the program to do the wrong thing. Debugging is an important skill for programmers who must be able to identify errors, find them in the program, correct them and then test that the bug is gone. More information on debugging (http://bit.ly/CSScot146)

Computing language / environment
Just like people, computers often understand a variety of different languages which are suitable for different purposes. Some computing languages are suitable for use by experts, or for certain sorts of scientific or data management tasks. Expert programmers are more likely to use textual languages which are written using English words and phrases with clearly specified rules (known as syntax and semantics). Beginner programmers will likely find a visual programming language easier to learn. In a visual language, the program is shown as a mixture of graphics and text, and the programmer can drag the graphics around to change the sequence of instructions. Often visual languages are blocks based, where programing constructs are represented in coloured blocks which look like jigsaw pieces. Scratch Jr is an example of an icon-based blocks language, as it uses blocks, icons and symbols so that programmers do not need to read or write. More information on computing languages (http://bit.ly/CSScot147)

Computational thinking
Computational thinking is a powerful approach to solving problems. It is commonly used in computer science, but it is applicable to many everyday problems too. It allows us to take a complex problem, understand the problem better by using a computational framework, and develop possible solutions. We can then present these solutions in a way that a computer, a human, or both, can understand.

Decomposition
Decomposition is when you solve a big problem by breaking it up into smaller bits, solving those, and sticking the smaller solutions together into a final answer. More information on [decomposition](http://bit.ly/CSScot149) (http://bit.ly/CSScot149)

Deterministic
An attribute of a process, which means that its outcome is predictable and repeatable. For example, the outcome of moving a counter a given number in Snakes and Ladders is deterministic, but the outcome of throwing the dice is not deterministic. All computing processes are in fact deterministic, unless their behaviour depends on non-deterministic input. Modern computing devices are so complex that this is not always obvious.

Event handling
It is common for a program to have to respond to input from the user, or messages from sensors or other computers. This is called event handling. For example, in a language like Scratch, there are blocks to help the programmer write code to respond to events like key presses from the user or the sprite colliding with another sprite.

Hidden mechanism
A mechanism normally hidden from the view of a user of a device, for example the engine of a car or the wash cycle of a washing machine. Users often need to know something about this mechanism in order to use the device effectively; for example a washing machine does not quite function as a “magic box” where clothes go in dirty and come out clean, although this is the abstraction the designers aim for. We can use the machine more effectively when we understand more about how it works, about temperature, spin speeds and timing of washing cycles. Similarly, the more that a user understands the hidden mechanisms in computing devices such as smartphones and the Internet, the more value they can get from them.

Information
Any kind of fact or knowledge about something tangible, like a physical item, or intangible, like an idea. In computing information is represented by a sequence of symbols which can be understood by a person or machine that will interpret the information. In computer memory all information is stored in the simplest form of information possible, binary.

Input / Process / Storage / Output
This is the sequence of activities computers generally perform. The computer takes in information as *input* (perhaps the user types on the keyboard or moves the mouse), *processes* that information by following a sequence of instructions from a computer program, *stores* the results of the program for later (on a hard disk) and then *output*s the results to the user (such as changing the display on the screen or playing a sound through the speaker). More information on computing systems (http://bit.ly/CSScot148), inputs (http://bit.ly/CSScot141) and outputs (http://bit.ly/CSScot142)

Interface
We use "interface" in this document to describe the part of a piece of software which the user sees and interacts with. In any software package, there is a lot of code which happens behind the scenes which the user has no need to be aware of. The interface between the program and the user enables the user to enter input (perhaps using a mouse or keyboard) and see the output (usually on screen). The way in which a computer interacts with people is sometimes known as its Human-Computer Interface.

Logic (AND / OR / NOT)
Computers are built to process instructions containing boolean logic. Key words to learn here are AND, OR and NOT. Of course, we use these words in everyday language, but it is worth checking that your learners understand their precise meaning. These concepts are often used within selection statements in programs when the computer should take different action depending on the information it is processing. More information on logic (http://bit.ly/CSScot150). Here are some examples:

- IF the username is correct AND the password is correct THEN display the home page. (both conditions need to be true with AND)
- IF the password is NOT correct THEN display an error message. (NOT is about checking whether the opposite of a statement is true)
- IF the number of lives gets to zero OR the timer gets to 60 THEN display the message "You lose!". (For OR, if either or both of the conditions are true, then the action should be taken.)

Mental models
In the context of computing education, we use the term "mental model" to describe a learner's understanding of how a computer system works. Novice programmers often have an incomplete and flawed model of how a program will execute which makes it harder for them to find bugs.

Parallel
The term "parallel" is used to describe two or more processes which occur at the same time. For example, in Scratch you can have more than one script running at a time such as a cat sprite chasing a dog sprite.

Process
A dynamic series of connected actions, many of them occurring one after the other in a sequential order, normally with apparent start and finish states (some processes may never finish however.) The behaviour of a process may depend on its context perhaps through input or observation of its environment. The behaviour of a process in a game might be affected by sensing the race car is driving over green grass, or a user clicking the mouse. The process of getting ready for play time might change if we see dark rain clouds outside.

Repetition

Computers excel at doing the same thing over and over again without making a mistake. Most programming languages make it easy for the programmer to instruct the computer to repeat tasks using loops. **Fixed repetition** is when the computer is instructed to carry out a sequence of steps a certain number of times, e.g. “do this ten times: move one step to the left”. **Conditional repetition** is when the program keeps repeating a sequence of steps until a condition becomes true e.g. “keep doing this: if you haven’t hit the side of the screen yet, move one step to the left”. More information on repetition (http://bit.ly/CSScot151)

Representation (pictorial, iconic, physical, etc)

Representations matter a lot in computer science, because sometimes problems are easier to solve if they are represented in a different way. For example, when young children learn about sequences, it might be easiest for them to understand the concept by interacting a physical robot. As they get better at understanding symbolic representations (such as pictures, diagrams or text), they can then reason about sequences more fluently because they do not have to rely on their memory. They can read a representation of a program and predict what it will do, or write their own instructions in pictorial form. Computers process on 1s and 0s, but this is an awful representation of information for humans because it is very hard to remember what long sequences of binary mean. This is why, behind the scenes, computers translate programs written in a language we understand into binary. Visual programming languages have a clearer representation for novices, but for experts they may be too cumbersome.

Searching

Computers spend a lot of time searching for specific items in large collections of information (for example, finding a customer name in a list of 80,000 customers). Because of this, computer scientists have spent a lot of time working out mathematically efficient ways to search information quickly. Often this requires the information to be carefully sorted to make searching easily. Simple search algorithms can also be used in real life problems.

Sequence

A series of actions, where the actions occur one after another in the order they are listed in. More information on sequence (http://bit.ly/CSScot152)

Selection

Making a choice about what action to carry out next based on testing if a condition is true or false. We can select whether to do something or nothing, select between two possible actions or select one of many possible actions Selection statements are often expressed in the format IF a condition is true THEN do something ELSE do something different. More information on selection (http://bit.ly/CSScot153)

State
A process or a program moves through a series of states as it executes instructions. A state is a useful abstraction to help you think about the main stages of a program and how the program moves between the states without worrying about the detail. For example, an order in an online shopping web site could be in the state of: order requested, purchased, delivered, or order complete. Every process should have one or more start or end states.

Sorting
Information isn't very useful if it is stored in a big jumble. Computer scientists like their information to be sorted to make it easier to find items later or to process it in other ways. Information which is clearly structured is easier to sort. You can sort the same collection of information in different ways depending on which attribute you use. For example, if you had a collection of information about pupils in your school, you could sort them according to age or height, or sort them by class and then surname.

Sprite
This is a term from animation, also used in some visual programming languages, to refer to a 2D picture which can move around the screen. In Scratch, sprites can have blocks attached to them which control their behaviour.

Predictable and non-predictable
In this document, we use the term "predictable" to describe programs where it is possible to look at the input and the program code and work out what the output will be. By "non-predictable" we mean programs where it is not possible to say in advance what the output will be given the input and the program code because the program uses randomness. For example, if you have a program which uses the equivalent of a dice roll to choose between six options, you know in advance the range of *possible* outputs, but you don't know exactly which one will happen in any given run of the program.

Programming constructs
Most programing languages share a set of common useful features such as selection (IF... ELSE) and repetition (fixed loops using REPEAT or FOR, conditional loops using FOREVER or WHILE), as well as ways of storing structured information. These are referred to as programming constructs. The constructs might look different in different languages, but they tend to work in a similar enough way that a programmer who knows one language can adapt to another one. You might also see the terms "control structures" or "control flow elements" to refer to programming constructs which specify what the program should do next in a sequence.

Unplugged Computing
You don't need a computer to learn about computing concepts! "Unplugged" computing is when you use computational thinking away from the computers, for example with physical games or pencil and paper.

Variables
A name given to an abstract concept within a computer program to store information temporarily while the program is running. Variables also exist in mathematics, for example the variable *pi* is used to refer to the ratio of a circle's circumference to its diameter, and variables such as *x* and *y* are used in equations to refer to numbers whose value is not yet known. Unlike in algebra, however, computing variables can change their value over time while a program is running, for example to store the score of a game or for a countdown timer. More information on variables (http://bit.ly/CSScot154)

The Scottish Curriculum: A brief guide for international readers

Children in Scotland start primary school at between age 4½ to 5½. They attend primary school for seven years (P1 to P7). Then aged eleven or twelve, they start secondary school for a compulsory four years (S1 to S4) with the following two years (S5 and S6) being optional.

The Scottish curriculum has two stages: the broad general education (from the early years to the end of S3) and the senior phase (S4 to S6).

The **broad general education (BGE)** has five levels:

- Early level - Early Years and Lower Primary
- First level - Lower Primary
- Second level - Upper Primary
- Third level - Lower Secondary
- Fourth level - Optional learning outcomes to challenge learners before they move on to the senior phase

The curriculum at the broad general education stage comprises of groups of individual learning outcomes, called Experiences and Outcomes.

- **Experiences and Outcomes** (Es and Os) are a set of clear and concise statements about children's learning and progression in each curriculum area. They are used to help plan learning and to assess progress throughout the broad general education.
- **Significant Aspects of Learning** (SALs) are overarching themes across groups of Experiences and Outcomes.
- **Benchmarks** set out clear statements about what learners need to know and be able to do to achieve a level in that curricular area.

The **senior phase** is designed to build on the experiences and outcomes of the broad general education, and to allow young people to take qualifications and courses that suit their abilities and interests. Learners study for qualifications at 'National 3', 'National 4' or 'National 5' in S4 (between the ages of fourteen to sixteen). After completing National 4/5s, learners may choose to stay at school and study for additional National qualifications, or progress on to Higher and/or Advanced Higher qualifications. Scottish Secondary schools have taught Computing Science as part of the senior phase since the 1980s.

More information:
https://education.gov.scot/scottish-education-system/
https://en.wikipedia.org/wiki/Education_in_Scotland

Web links

The web links used throughout the document are listed below for those accessing the document in a printed form.

SAL 1 Early Level (Page 22)
Classifying objects activities - http://bit.ly/CSScot1
OK Go: This Too Shall Pass - http://bit.ly/CSScot2

SAL 3 Early Level (Page 24)
Beebots - http://bit.ly/CSScot3
Ozobot - http://bit.ly/CSScot4
Sphero - http://bit.ly/CSScot5
SPRK - http://bit.ly/CSScot6
Ollie - http://bit.ly/CSScot7
Dash and Dot - http://bit.ly/CSScot8
Lightning Lab - http://bit.ly/CSScot9
Tynker - http://bit.ly/CSScot10
Scratch - http://bit.ly/CSScot11
Beebot web-based emulator - http://bit.ly/CSScot12
Bits and Bytes - http://bit.ly/CSScot13
Robot Turtle - http://bit.ly/CSScot14

SAL 2 First Level (Page 26)
ScratchJr - http://bit.ly/CSScot15

SAL 3 First Level (Page 27)
Hour of Code course 1 (for 4-6 year olds) - http://bit.ly/CSScot16
Cargobot - http://bit.ly/CSScot17
Robot School - http://bit.ly/CSScot18
Move the Turtle - http://bit.ly/CSScot19
The Foos - http://bit.ly/CSScot20
Kodable - http://bit.ly/CSScot21
Daisy the Dinosaur - http://bit.ly/CSScot22
Lego WeDo - http://bit.ly/CSScot23
Create your own programming board game - http://bit.ly/CSScot24

SAL 1 Second Level (Page 28)
Lego bubble sort - http://bit.ly/CSScot25
Bubble sort song - http://bit.ly/CSScot26
Robots quick sort - http://bit.ly/CSScot27

SAL 2 Second Level (page 30)
CS Unplugged - http://bit.ly/CSScot39
Teach London Computing - http://bit.ly/CSScot40
A Little Bit of CS4Fn - http://bit.ly/CSScot41
Barefoot Computing - http://bit.ly/CSScot42
Google's Exploring Computational Thinking - http://bit.ly/CSScot43
Starting From Scratch - http://bit.ly/CSScot44

SAL 3 Second Level (Page 31-32)
Kodu - http://bit.ly/CSScot28
Hopscotch - http://bit.ly/CSScot29
Minecraft - http://bit.ly/CSScot30
Mozilla Thimble - http://bit.ly/CSScot31
App Inventor - http://bit.ly/CSScot32
Lego Mindstorms - http://bit.ly/CSScot33
First Lego League - http://bit.ly/CSScot34
Vex Robotics - http://bit.ly/CSScot35
Vex competitions - http://bit.ly/CSScot36
Makey Makey - http://bit.ly/CSScot37
Microbit - http://bit.ly/CSScot38

Photo Credits

All photos used in this document have been used with kind permission by the copyright owner or are licenced under Creative Commons. To reuse these images, please refer to the licence terms on using the links below or contact the copyright owner directly.

Google car by smoothgroover22, https://www.flickr.com/photos/smoothgroover22/15104006386/
Code Kingdoms code by Code Kingdoms, https://codekingdoms.com/graphics/articles/article-editor.ckh.77b04e7.gif
Javascript code by Dmitry Baranovskiy, https://www.flickr.com/photos/dmitry-baranovskiy/2378867408/
Woman on computer by the European Parliament, https://www.flickr.com/photos/european_parliament/8704362988/
Bee-bots cards by www.ocg.ac https://www.flickr.com/photos/ocg_bilder/8571558494/
ScratchJr cards in action by Tim Ley @tfley, http://libraryadventuring.blogspot.co.uk/2015/11/why-my-k-2-students-will-be-using.html
Scratch cards in action by Bobby Lewis @usabbs, https://twitter.com/usabbs/status/746708479554035712
Beebot in action by Ars Electronica Center, https://www.flickr.com/photos/arselectronicacenter/24863185599/ The Foos tablet app by Code Spark (press kit), http://thefoos.com/coding-resources-for-you/
Water wall play by Cathy Denman, Pre-school Play, http://pre-schoolplay.blogspot.co.uk/2011/05/water-wall.html
Paper airplanes by Bernardo R, https://www.flickr.com/photos/bernardoramonfaur/24907446110
Snakes and Ladders by Jacqui Brown, https://www.flickr.com/photos/120600995@N07/14125947172
Pokemon cards by Jinx! https://www.flickr.com/photos/span112/3602277316
Sticky note arrow commands and DIY board game by Sarah McClelland, http://littlebinsforlittlehands.com/superhero-computer-coding-game-without-a-computer/
Marbles by Dale Jackson, https://www.flickr.com/photos/stratoart/8928459014/
Occulus rift by Southbank Centre, https://www.flickr.com/photos/southbankcentre/18678970371
Scratch vector blocks by Instituto Thomas Jefferson, Querétaro, México http://scratched.gse.harvard.edu/stories/scratch-hands-blocks-itj-early-start-through-physical-interaction
Scratch command dice by Adriano Parracciani, http://scratched.gse.harvard.edu/resources/dice-4-scratch
The Foos screenshot by Code Spark (press kit), http://thefoos.com/coding-resources-for-you/
Ozobot spiral example by Diane Cordell, https://www.flickr.com/photos/dmcordell/17287880645
Lego Mindstorms by Brandan Newendorp, https://www.flickr.com/photos/bnewendorp/3459925450/
Makey Makey plasticine controller by Zatna LLC, https://www.flickr.com/photos/97687927@N02/18683404733/
BBC Microbit by Gareth Halfacree, https://www.flickr.com/photos/120586634@N05/26146391212/
Scratch racing game by 12dengb, https://scratch.mit.edu/projects/673918/
Playground bitmap images by CS Unplugged, http://csunplugged.org/image-representation/
Sphero SPRK robots drawing shapes with paint, https://brandfolder.com/SPRKedu
Scratch cat by the Lifelong Kindergarten Group at the MIT Media Lab via Wikimedia Commons https://commons.wikimedia.org/wiki/File:Scratchcat.svg
Barefoot Computing concept icons by Computing at School / Department for Education http://barefootcas.org.uk/barefoot-primary-computing-resources/concepts/