

Teach Computing Science

A Guide for Secondary Practitioners

Kate Farrell, Computing at School Scotland

Tom Hendry, Balerno High School

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. Curriculum Organisers were introduced as overarching themes across groups of Es and Os, and Benchmarks were added as examples of typical activities.

This guide introduces and explains the Computing Science (CS) Organisers and the updated experiences and outcomes and Benchmarks. It provides an exemplification guide and resources for use in Secondary schools. 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 grouping of computer science concepts according to three Curriculum Organisers. The first Organiser introduces learners to core concepts in CS. The second Organiser introduces learners to how tools and languages use those concepts. The third Organiser sees learners apply their learning from the first two Organisers 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 Organiser 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 the Third and Fourth levels are covered in detail here, suitable for learners who have worked within this framework at primary school as well as those who have little experience. We have a separate guide available for Early Years and Primary practitioners.
- 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.

Curriculum Organisers for Computing Science

Curriculum Organisers for Computing Science

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

Organiser 1: Understanding the world through computational thinking

This Organiser 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.

Organiser 2: Understanding and analysing computing technology

This Organiser 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.

Organiser 3: Designing, building and testing computing solutions

This Organiser is about: Creating
Use conceptual and technological knowledge to design, build and test.

The Curriculum Organisers for Computing Science 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 Organisers 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 (Organiser 3), we need to understand precisely how computers are told to do anything at all (Organiser 2) - and to understand that, we need to know what kinds of tasks computers can carry out (Organiser 1)



The Curriculum Organisers for CS build upon each other. The conceptual knowledge gained when working towards the first organiser, '**understanding the world through computational thinking**', is required to then **understand computing languages and technologies** in the second organiser, before we can then '**design, build and test computing solutions**' in the final organiser using those technologies.

It is expected that learners will be able to understand more complicated concepts in the first organiser than they are capable of reading or writing themselves in the second and third organisers. 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 (Organiser 1), languages and tools (Organiser 2) before going on to develop and build computing solutions (Organiser 3). As the organisers complement each other, it is expected that more than one organiser 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 (Organiser 3) without knowledge and understanding of the underlying concepts (Organiser 1) or being able to accurately read and understand programs in that language (Organiser 2).

Definitions of the Curriculum Organisers in Computing Science

Understanding the world through computational thinking

The first Curriculum Organiser 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 finding the shortest or fastest routes, most efficient methods of doing a task, or classifying objects.

Learners will be able to identify and analyse steps and patterns in a **process**, for example following the steps in a Hungarian algorithmic folk dance or singing along to a flowchart of the lines of a song. Learners will reason about properties of more complex processes, for example considering whether tasks could be carried out at the same time are more efficient when they communicate, or using logic to (AND, OR, NOT) to deduce facts from statements in logic puzzles.

Learners will classify **information**, sort it in different ways and look for ways to search the information in a fast or efficient manner. For example, groups of learners might race to sort Top Trumps cards using different sorting algorithms.

Understanding and analysing computing technology

This Curriculum Organiser 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 Organiser learners will learn how to 'read' program code (before writing it in the next Organiser) and describe its behaviour in terms of the **processes** they have learned about in the first Organiser, 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.

This Organiser 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 Curriculum Organiser is about taking the concepts and understanding from the first two Organisers and applying them. Learners will create solutions, perhaps by designing, building and testing solutions on a computer. 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, data models, databases and web pages 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 which might not necessarily be what they intended, as well as giving them practice in debugging.

Themes across the Curriculum Organisers: Information and Process

Running through the three Curriculum Organisers 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 Organiser.



```
public class Player extends BasePlayer {  
    public void onPlaceBlock(Block block) {  
        world.strikeLightning(block.getLocation());  
        canMine = true;  
    }  
}
```

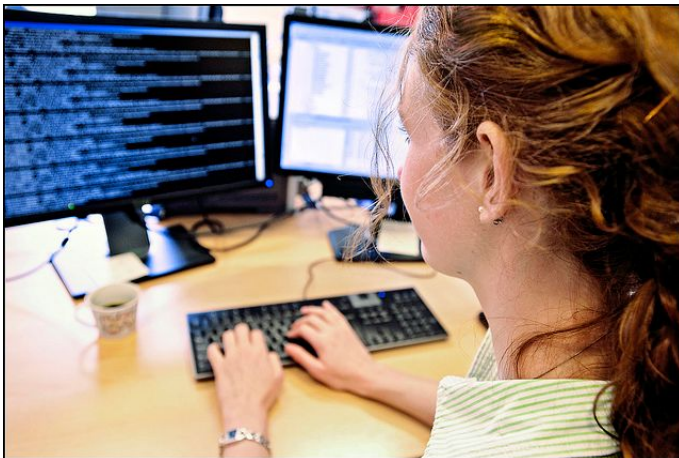
A computing system does not spontaneously decide what process to carry out, or what information to manipulate. It is told precisely what to do via a set of instructions held in a computer program. These instructions are written in a programming language. Such languages are not at all like the sort of 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 our 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 Organiser.

The third Organiser 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 Organisers can be just as much fun!) Organiser 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 (Organiser 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 (Organiser 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.

Computing Science

Progression of Concepts

Core Computing Science Concepts across the Organisers

As noted earlier, the Organisers 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 Organisers.

Concepts	Early Level	First Level	Second Level	Third Level	Fourth Level
Structuring Processes	Sequence (of movements)	Simple sequences Selection - sequences with conditional statements	Single or parallel sequences Variables	Parallel sequences that interact or communicate Multiple variables	Finding efficient and optimal processes. Processes involving structured information such as lists or arrays
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	More complex conditional statements, including logic (AND, OR, NOT). More complex repetition such as nested loops.	Describing processes in the real world and comparing alternative solutions
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	Comparing different sorting and searching algorithms Sourcing, querying, structuring and manipulating information such as flat file databases, program or webpage.	Compare implementations of different sorting and searching algorithms for efficiency Sourcing, querying, structuring, linking and manipulating information in relational databases.
Computing Systems	Computers follow instructions	Computers take in inputs, process them, store information, and then output the results	Computers can communicate over networks	Computers use low level language to represent information. Information can be compressed and encrypted.	Understand different file formats are suitable for different types of information Understanding how computing is used in modern technologies

Progression of Information concepts

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.

Third level:

Learners will be manipulating information using more complex conditions and logic. They could try out and compare different sorting and searching algorithms, perhaps sorting cards or books in the classroom.

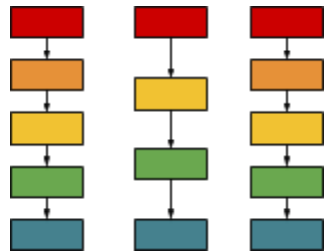
Learners should gain experience in sourcing, structuring, querying and manipulating information by using and creating a flat file databases, program or webpage. Currently, visual languages such as Scratch do usually provide limited functionality for variables and lists, although it might be complicated to manipulate these lists in any meaningful way. It may be simpler for learners to explore these concepts in a simple database program such as Filemaker, data analysis package such as Tableau or CODAP, or a table in an HTML web page.

Fourth level:

Learners can compare implementations of different sorting and searching algorithms for efficiency. Learners could time how long it takes a program to sort a large amount of data using a quick sort algorithm compared to a bubble sort. Learners could be then be involved in sourcing, querying structuring, linking and manipulating information in relational databases or in a data analysis package. For example, they can query existing open source data sets to discover how air quality varies across a city, and create their own databases with tables for actors and movie titles.

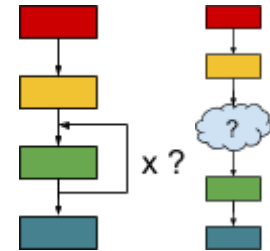


Progression of Process concepts



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. 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).



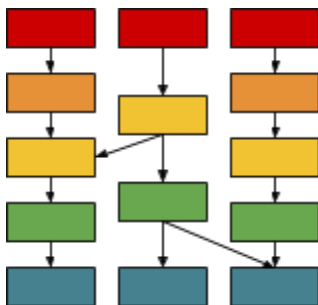
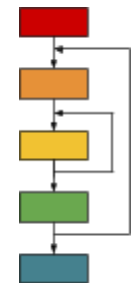
Learners can see patterns in problem solving and identify a solution that has been used previously. For example, when creating a set of instructions on how to get to the head teacher, they can reuse instructions on how to get to the school office, or instructions on how to cook pasta can be adapted for boiling rice. Later, if learners are creating games in Organiser 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.

Third level

Learners use more complex conditional statements, including logic (AND, OR, NOT), for example loop the racing car instructions while remaining time is greater than one and the car is not touching the red finish line.

Learners can identify when a complex conditional loop will stop or a complex conditional statement will start. Learners can identify and make use of complex repetition such as nested loops (for example drawing shapes multiple times to create spirograph-style images).

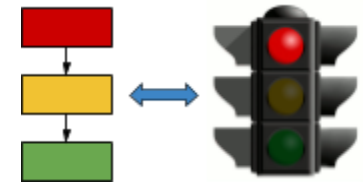


Learners will use continue to use parallel sequences such as multiple sprites, but learn that they can interact or communicate. This might be a sprite that senses if it is interacting or touching another sprite, or using broadcast-style commands to communicate with all sprites.

Learners understand that you can keep track of more than one variable, such as keeping track of both time and score in sports. They can then learn about code for creating and using multiple variables in a visual language, including using conditional statements to make decisions based on the state of these variables, such as the game ends if health or time gets to zero.

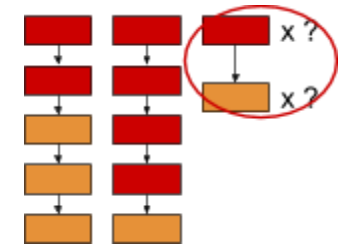
Fourth level

Learners are able to spot and describe processes in the real world such as airports, parcel delivery systems, transport systems, self-driving cars or farming. They can look at an existing system and reason about rules, heuristics and processes involved in the system. They could also think about ways to improve these systems.



Learners can identify systems in the real world that collect data and use it to make decisions. For example, learners could look at their data from a step counter and analyse it in order to improve their health and fitness, or use live tracking travel data to plan a journey, perhaps involving more than one method of transport.

In Organiser 2, learners could compare alternative solutions and discuss the benefits and drawbacks of different approaches in terms of efficiency. They could compare solutions that they are given, or each learner could plan their own solution to a problem and then compare with others in the class to find out which solution would be fastest or involve the least number of instructions.



This might involve learners trying to find optimal processes, for example they could use accelerometer data from a Microbit to measure their individual gait or arm swing as they walk. They could then try to program a Microbit as a step counter and compare results with other learners to come up with a more effective algorithm. Alternatively learners could design a game controller using a Makey Makey and conductive material such as tinfoil. Then they can work out which blocks in Scratch they will use with their controller.

Learners could also explore processes involving structured information such as lists or arrays. Learners would need to explore the concepts involved in data structures in Organiser 1, perhaps in an unplugged way, then moving on to Organiser 2 to learn about the commands they will need to use to implement a data structure and how they can be used in processes.

Progression of Languages

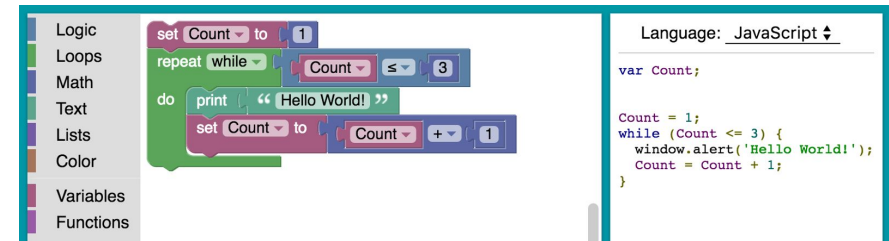
Second level:

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 the values stored in 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. 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.



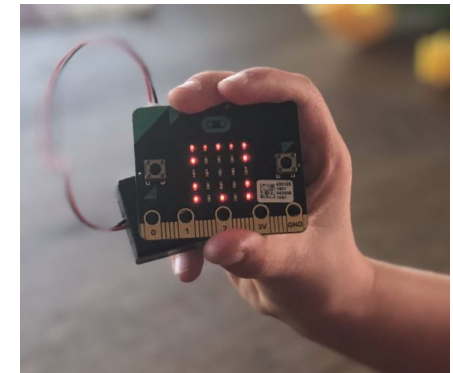
Third level:

At third level, learners may still be using the same visual languages as they used in Second level (such as Scratch) but they will be doing more complex operations in that environment. Languages related to Scratch, such as 'Snap!' might be useful due to their extended functionality but still being a familiar environment for learners who have used Scratch. It would also be beneficial for them to encounter other visual languages, such as Blockly, PencilCode or the Microbit Javascript Blocks editor, particularly those that allow the learner to switch between textual and visual/blocks modes.



Learners may be starting to look at textual languages. They will perhaps be using HTML and CSS for creating web pages, which will give them a context to learn about and explore syntax errors, but it might be useful for more secure learners to investigate the textual version of their code if using a transition environment such as Blockly, Microbit or PencilCode.

Learners will experience using multiple and mixed structures, such as combining multiple loops to draw spirograph shapes, or multiple conditions, for example creating a Rock Paper Scissors program on a Microbit. They should have an opportunity to try more complex logical conditional statements, such as *repeat while not finished*, or *if touching red but not positioned at the edge of the stage*. They could try following complex programs and predicting the output. Controlling physical devices, such as Microbit, Makey Makey or Raspberry Pi, will give learners the opportunity to experience more complex tasks with an engaging and rewarding output.



Fourth level:

Learners will be working in transition languages (that allow learners to switch between blocks and textual commands), plus trying a purely textual language. Confident learners should be progressing into textual languages by the end of Fourth level. In web languages, Fourth level learners will be more confident at HTML and CSS, and trying to add interactivity to their pages using simple lines of JavaScript.

However, it is still essential for learners to learn about the commands available in a textual language (Organiser 2) and be able to associate those with the concepts they learned in Organiser 1 and with blocks they have used in a visual language that carried out the same operation. It might be beneficial for learners to create a translation dictionary as they learn a new textual language, showing blocks, text commands and the outcome of running that block/command.

Computing Science

Experiences and Outcomes and Benchmarks

Organiser 1: Understanding the world through computational thinking

Experiences and Outcomes				
Early Level	First Level	Second Level	Third Level	Fourth Level
<p>I can explore computational thinking processes involved in a variety of everyday tasks and can identify patterns in objects or information.</p> <p style="text-align: right;">TCH 0-13a</p>	<p>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.</p> <p style="text-align: right;">TCH 1-13a</p>	<p>I understand the operation of a process and its outcome. I can structure related items of information.</p> <p style="text-align: right;">TCH 2-13a</p>	<p>I can describe different fundamental information processes and how they communicate, and can identify their use in solving different problems</p> <p style="text-align: right;">TCH 3-13a</p> <p>I am developing my understanding of information and can use an information model to describe particular aspects of a real world system</p> <p style="text-align: right;">TCH 3-13b</p>	<p>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</p> <p style="text-align: right;">TCH 4-13a</p> <p>I can informally compare algorithms for correctness and efficiency</p> <p style="text-align: right;">TCH 4-13b</p>
Benchmarks				
<ul style="list-style-type: none"> 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) 	<ul style="list-style-type: none"> 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) 	<ul style="list-style-type: none"> 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 	<ul style="list-style-type: none"> 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. 	<ul style="list-style-type: none"> 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?

Organiser 2: Understanding and Analysing Computing Technology

Experiences and Outcomes				
Early Level	First Level	Second Level	Third Level	Fourth Level
<p>I understand that sequences of instructions are used to control computing technology TCH 0-14a</p> <p>I can experiment with and identify uses of a range of computing technology in the world around me. TCH 0-14b</p>	<p>I understand the instructions of a visual programming language and can predict the outcome of a program written using the language. TCH 1-14a</p> <p>I can understand how computers process information. TCH 1-14b</p>	<p>I can explain core programming language concepts in appropriate technical language TCH 2-14a</p> <p>I understand how information is stored and how key components of computing technology connect and interact through networks. TCH 2-14b</p>	<p>I understand language constructs for representing structured information TCH 3-14a</p> <p>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</p>	<p>I understand constructs and data structures in a textual programming language TCH 4-14a</p> <p>I can explain the overall operation and architecture of a digitally created solution TCH 4-14b</p> <p>I understand the relationship between high level language and the operation of computer TCH 4-14c</p>
Benchmarks				
<ul style="list-style-type: none"> ● 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) 	<ul style="list-style-type: none"> ● 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. 	<ul style="list-style-type: none"> ● 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 	<ul style="list-style-type: none"> ● 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 	<ul style="list-style-type: none"> ● 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 between different levels of abstraction

Organiser 3: Designing, building and testing computing solutions

Experiences and Outcomes				
Early Level	First Level	Second Level	Third Level	Fourth Level
<p>I can develop a sequence of instructions and run them using programmable devices or equivalent</p> <p style="text-align: center;">TCH 0-15a</p>	<p>I can demonstrate a range of basic problem solving skills by building simple programs to carry out a given task, using an appropriate language.</p> <p style="text-align: center;">TCH 1-15a</p>	<p>I can create, develop and evaluate computing solutions in response to a design challenge.</p> <p style="text-align: center;">TCH 2-15a</p>	<p>I can select appropriate development tools to design, build, evaluate and refine computing solutions based on requirements.</p> <p style="text-align: center;">TCH 3-15a</p>	<p>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.</p> <p style="text-align: center;">TCH 4-15a</p>
Benchmarks				
<ul style="list-style-type: none"> • 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 	<ul style="list-style-type: none"> • 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 	<ul style="list-style-type: none"> • 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 	<ul style="list-style-type: none"> • 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 	<ul style="list-style-type: none"> • 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.

Planning Learning and Assessment Using the Organisers and Benchmarks

Understanding the Benchmarks for Computing Science

Each of the three Curriculum Organisers has a list of Benchmarks. These are examples of the standard expected to be successful, activities that learners can do in order to demonstrate they have experienced and achieved a level of competence in the outcomes within that Organiser.

The benchmarks are **not** prerequisites. Learners do not need to do them all. Teachers may choose to approach the outcomes in a different way and provide alternative experiences for learners. The FAQ paper on Benchmarks on the National Improvement Hub (Education Scotland), states

“In order to achieve a CfE level, it is not necessary for learners to demonstrate mastery of every individual aspect of learning in the Benchmarks for that level. However, it is important that there are no major gaps in children’s and young people’s learning when looking across the major organisers in each curriculum area.”

Education Scotland, <http://bit.ly/CSScot189>

In a changing environment such as Computing Science, the teacher needs to use their professional expertise to provide suitable challenges to learners. Perhaps in the future, a task that was previously challenging may become trivially easy using a particular development environment, language or tool. In this case, the teacher should think about what experiences would be suitable for Level 3 and 4 learners using the tools available to them at the time in order to experience the Computing Science Outcomes.

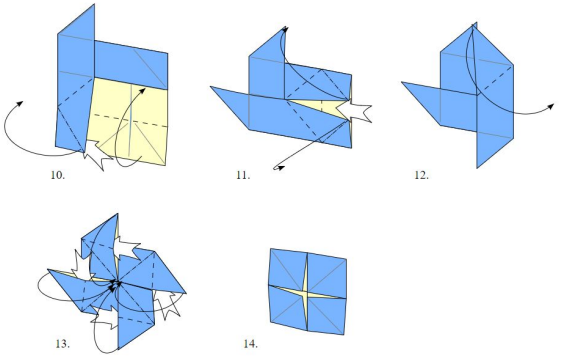
It is important to ensure that the philosophy of the Organisers is followed through, when approaching a particular concept. In order to tackle a particular challenge, learners must first understand the computing concepts involved (as they would do in Organiser 1), and learn about the tools available to them (as they would do in Organiser 2) before designing and creating their own solution to the challenge (in Organiser 3).

In order for learners have a good experience of an Outcome, they must first gain deep and secure understanding of a concept and then develop their knowledge of the tools required to implement that concept.

An activity might cover just one outcome or organiser. However, some activities or larger practical projects may provide an opportunity to assess a learner’s achievement of more than one outcome or organiser. With the spiral nature of the curriculum organisers, a learner may carry out an activity that assesses more than one outcome, where they can demonstrate an understanding of an underlying CS concept (Organiser 1) as well as an understanding of the tools and languages (Organiser 2) that they then use to create a computer solution (Organiser 3) As the organisers complement each other, it is expected that more than one organiser could be covered in a single lesson and learners revisit concepts at increasing depth as they work through Level 3 and 4.



Teaching and Assessment with the Benchmarks

Organiser 1 Third Level	Understanding the world through computational thinking
<p>Outcome:</p>	<p>Benchmarks:</p>
<p>I can describe different fundamental information processes and how they communicate, and can identify their use in solving different problems</p> <p style="text-align: right;">TCH 3-13a</p> <p>I am developing my understanding of information and can use an information model to describe particular aspects of a real world system</p> <p style="text-align: right;">TCH 3-13b</p>	<ul style="list-style-type: none"> • 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.
<p>What learning and assessment may look like:</p>	
<p>Recognises and describes information systems with communicating processes which occur in the world around me</p> <p>Learners should be able to give examples of paper-based and computer based information systems. They should be able to suggest the kind of information stored in those databases and explain how that information is normally ordered. They should understand that most websites are constructed from multiple databases, and that different users may see different content on that webpage. For example, different users logging into social media websites will see different listings of posts pulled from the service’s databases, and different people viewing a website with adverts will see different adverts depending on their web traffic.</p> <p>Explains the difference between parallel processes and those that communicate with each other</p> <p>Learners may have come across this concept in Scratch, where sprites have separate scripts but can communicate using the Broadcast block or through interaction (e.g. the ‘If touching sprite...’ block). Scripts can be parallel but independent of each other, all running when the flag is clicked, or they can rely on each other with some processes that only run when they receive a signal from another sprite.</p> <p>In order to teach this concept, learners could compare following instructions for a repetitive task where they either complete the whole process individually compared to groups working together, factory production line style, with each team member working on their part of the production.</p> <p>This could be simple origami instructions, with each learner in a team doing their part of the instructions before passing on to the next person. A team could create collage pictures by following cutting out and assembling different coloured shapes, or they could cut out, fold and assemble papercraft shapes from nets. A team member can’t continue until they have received the paper/information from another teammate (or a go-ahead flag). The team effort could be compared to the speed of individuals doing the whole task separately. This could also lead to a discussion about robotics in factories.</p> <div style="text-align: right;">  </div>	

Demonstrates an understanding of the basic principles of compression and encryption of information

Learners can explore different simple encryption methods such as Caesar cipher, pigpen, morse code (perhaps using a murder mystery theme such as <http://bit.ly/CSScot156>). Compression can also be taught in an unplugged way, such as activities from CSUnplugged (<http://bit.ly/CSScot157>). Learners could find out more about the Enigma machine and the history of Bletchley Park.



Identifies a set of characteristics describing a collection of related items that enable each item to be individually identified

This benchmark helps learners start to think about classification of objects and thinking about characteristics of that object. Learners in groups could pick a topic and select a number of associated items (such as different animals, cartoon characters or films). They should then think of ways to describe the items, to differentiate them. Discuss the need for unique identifiers in some situations, so that objects, items and people can be selected without confusion. Learners could turn this information into a trading card game.

An example of identifying based on characteristics might be tables represented with different characteristics for different purposes. Information about tables being sold on a website by a furniture company will need to include the dimensions of the object as well as characteristics such as weight, material and price. We need to be able to uniquely identify the table so that the customer can collect the right box from the warehouse, or have the correct product delivered to their home.

Information about tables in a restaurant computer system might be represented by a map displaying where they are located, identifying the waiting staff responsible for that table the number of patrons able to sit there. This might link it to a list of food consumed and the price of that food, in order to produce a bill. We need to be able to uniquely identify the table so that the customers don't end up with the wrong bill.

Identifies the use of common algorithms such as sorting and searching as part of larger processes

Learners could race to sort and search for information using different algorithms. There are many unplugged activities for teaching sorting networks (<http://bit.ly/CSScot158> and <http://bit.ly/CSScot169>), sorting algorithms (<http://bit.ly/CSScot159>), bubble sort (<http://bit.ly/CSScot160>) and merge sort magic (<http://bit.ly/CSScot161>). Unplugged activities for searching include algorithms (<http://bit.ly/CSScot162>), 20 questions (<http://bit.ly/CSScot163>) and even magic tricks (<http://bit.ly/CSScot164>).

Organiser 2 Third Level	Understanding and analysing computing technology																																															
Outcome:	Benchmarks:																																															
<p>I understand language constructs for representing structured information TCH 3-14a</p> <p>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</p>	<ul style="list-style-type: none"> • 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 																																															
What learning and assessment may look like:																																																
<p>Understands that the same information could be represented in more than one representational system We can show information such as the current time in different ways, such as using a digital clock, and analogue clock, or even using a sundial. We can show our age in years, work out how many days old we are, or even work out our age in Martian years.</p> <p>In the classroom, learners could learn how to use an ASCII conversion table to show their name in binary (perhaps even representing the binary as a bracelet using different coloured beads). Learners could encrypt messages from English into Morse code or into Caesar cipher using a code wheel.</p> <p>Discuss how to represent colours in a computer. When learning about HTML and CSS, learners can find out different ways to represent colours, such as HEX codes and RGB. Learners could use a website such as https://color.adobe.com/ to choose a colour scheme for a page. Ideally they should be able to work out RGB codes for simple colours (red, green, blue and colours mixing two primary colours, such as purple = red + blue) without a chart or web reference.</p> <p>Understands that different information could be represented in exactly the same representation We can use binary to store textual information but also images, sounds and videos. In class, learners could use an unplugged activity to create simple black and white binary images and convert them into binary code (http://bit.ly/CSScot157). Learners can swap their code with others and try to recreate the images.</p> <table border="1" data-bbox="1742 852 2056 1198"> <thead> <tr> <th rowspan="2">HTML name</th> <th colspan="3">R G B</th> </tr> <tr> <th>Hex</th> <th colspan="2">Decimal</th> </tr> </thead> <tbody> <tr> <td colspan="4">Purple, violet, and magenta colors</td> </tr> <tr> <td>Lavender</td> <td>E6 E6 FA</td> <td>230</td> <td>230 250</td> </tr> <tr> <td>Thistle</td> <td>D8 BF DB</td> <td>216</td> <td>191 216</td> </tr> <tr> <td>Plum</td> <td>DD A0 DD</td> <td>221</td> <td>160 221</td> </tr> <tr> <td>Violet</td> <td>EE 82 EE</td> <td>238</td> <td>130 238</td> </tr> <tr> <td>Orchid</td> <td>DA 70 D6</td> <td>218</td> <td>112 214</td> </tr> <tr> <td>Fuchsia</td> <td>FF 00 FF</td> <td>255</td> <td>0 255</td> </tr> <tr> <td>Magenta</td> <td>FF 00 FF</td> <td>255</td> <td>0 255</td> </tr> <tr> <td>MediumOrchid</td> <td>BA 55 D3</td> <td>186</td> <td>85 211</td> </tr> <tr> <td>MediumPurple</td> <td>93 70 DB</td> <td>147</td> <td>112 219</td> </tr> </tbody> </table>		HTML name	R G B			Hex	Decimal		Purple, violet, and magenta colors				Lavender	E6 E6 FA	230	230 250	Thistle	D8 BF DB	216	191 216	Plum	DD A0 DD	221	160 221	Violet	EE 82 EE	238	130 238	Orchid	DA 70 D6	218	112 214	Fuchsia	FF 00 FF	255	0 255	Magenta	FF 00 FF	255	0 255	MediumOrchid	BA 55 D3	186	85 211	MediumPurple	93 70 DB	147	112 219
HTML name	R G B																																															
	Hex	Decimal																																														
Purple, violet, and magenta colors																																																
Lavender	E6 E6 FA	230	230 250																																													
Thistle	D8 BF DB	216	191 216																																													
Plum	DD A0 DD	221	160 221																																													
Violet	EE 82 EE	238	130 238																																													
Orchid	DA 70 D6	218	112 214																																													
Fuchsia	FF 00 FF	255	0 255																																													
Magenta	FF 00 FF	255	0 255																																													
MediumOrchid	BA 55 D3	186	85 211																																													
MediumPurple	93 70 DB	147	112 219																																													

Demonstrates an understanding of structured information in programs, databases or webpages

Before creating a webpage (as part of Organiser 3), learners first need to know about different ways to structure information on web pages such as different sized headings, paragraphs, bullet point lists, images and links. Learners could draw a plan showing where information will be on a page and also the structure of the information, for example the tags they might use. Learners could view a webpage using Mozilla's X-ray goggles (<http://bit.ly/CSScot199>) and make changes to the page, for example changing a news story to be about themselves and their friends.

When planning a database, learners should think about the fields, the field types and the minimum and maximum values. It would be useful to look at Top Trumps cards or other similar games, to think about the fields and ranges of values.

When planning programs learners need to think about the variables their programs will need and names for them. Learners will also need to consider the order in which programs processes this information while planning programs.

Describes the effect of markup language on the appearance of a webpage, and understand that this may be different on different devices

Use a website such as CSS Zen Garden to demonstrate how different webpages can look with different markup CSS code. When creating their own HTML and CSS code, learners can see how the page should appear on phones. They can publish their page and try viewing it on their phone or at home. Learners could view web pages in different available browsers, to see if there are differences.

Demonstrates an understanding of the von Neumann architecture and how machine code instructions are stored and executed within a computer system

Learners should be able to identify and classify different input, output and storage devices. Learners should understand how information flows using a Von Neumann model diagram (Input, Process, Output, Storage). Discuss with the learners what happens when a key is pressed on a keyboard, that a signal is sent in binary using the ASCII code they might be familiar with from the first Organiser. The computer then processes that according to the operating system and software instructions, and then a character should appear on the screen.

Learners could try a Little Man Computer simulator to gain an appreciation of how a computer uses machine code instructions to control every operation.

Reads and explains code extracts including those with variables and data structures

Learners should be able to look at code (such as a screenshot of a Scratch or Microbit program) and predict what will happen when the program runs. They should be able to talk through what will happen on each line of code under different conditions (for example, what will happen to the 'Score' variable when Scratch code runs and the sprite is touching the edge, or over another sprite. Learners could also look at a program and try to work out the blocks or code that has been used to create the effect or output.

Demonstrate an understanding of how computers communicate and share information over networks

Simple network protocols, including the concepts of sender, receiver, address and packets, could be taught through an unplugged activity such as Tablets of Stone (<http://bit.ly/CSScot165>) and Routing and Deadlock (<http://bit.ly/CSScot166>). Discuss with learners how a webpage can be assembled from elements from different locations (eg fonts from Google fonts, images loaded from Flickr, videos coming from YouTube) and each element needs to be requested and addressed separately.

Cisco have useful tools and resources available at their Network Academy for schools (<http://bit.ly/CSScot175>) including a useful packet tracer tool (<http://bit.ly/CSScot176>).

Learners can design their own network protocol using the radio feature of the Microbit. They should consider what channel the protocol will work on, what values and messages will be sent, and what will they mean to the receiver.

Understands simple compression and encryption techniques used in computing technology

After exploring simple encryption methods (in Organiser 1) such as Caesar cipher, pigpen, morse code, learners can then discuss how easy these are to break now, and the methods used currently such as public and private keys. This could be done using an unplugged activity on public key encryption (<http://bit.ly/CSScot167>). After learners have explored compression in an unplugged way (in Organiser 1), they can try practical compression activities such as creating an image or recording a sound clip and saving it using a compressed file format and seeing/hearing the difference between uncompressed and compressed files.

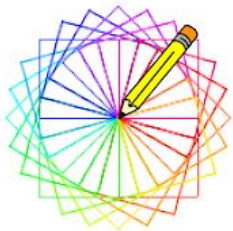
Organiser 3 Third Level	Designing, building and testing computing solutions
Outcome:	Benchmarks:
<p>I can select appropriate development tools to design, build, evaluate and refine computing solutions based on requirements.</p> <p>TCH 3-15a</p>	<ul style="list-style-type: none"> • Designs and builds 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
What learning and assessment may look like:	
<p>Designs and builds a program using a visual language combining constructs and using multiple variables</p> <p>At Third level, learners should be expanding their use of visual languages and doing more challenging activities in environments such as Scratch. They should be able to create games using variables for the score of opposing players, as a timer, or to keep track of lives or health points. Learners should be able to cope with using more complex constructs than they might have used at Second level, such as conditional repetition with ‘Repeat...until’ or ‘If...then...else’ for selection.</p> <p>An S1 learner might create a simple maze game with a simple ‘If’ statement to check if a sprite is touching a colour, or a single-level platform game. An S2 learner might create a more complex game with nested ‘If’ statements checking for multiple conditions, or a scrolling background platform game using multiple sprites as background tiles.</p> <p>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</p> <p>Learners could use a database program such as Filemaker Pro or Access to store information on a topic they are interested in. For example, they could create a ‘Top Trumps’ style database on superheroes or sports teams, search for the information to put into the database and design an attractive layout to look like a trading card game. Learners could also use a List structure in Scratch to keep track of players names and high scores.</p> <p>Interprets a problem statement, and identifies processes and information to create a physical computing and/or software solution</p> <p>Learners should be able to break a problem down into solvable pieces. For a database, they should be able work out suitable field names, appropriate field types and then identify information to enter into the fields.</p>	

For a Scratch game, learners should be able to identify parts of a problem that they already know how to solve (such as making a sprite move using the arrow keys or sense if it is over a particular colour and move back). They should be able to identify processes for each of the components (the sprites and the stage).

After learning about how to do radio commands on a Microbit, learners could work together in pairs to create programs that do different things depending on the radio input from their partner's Microbit. They would need to agree the information that will be sent and received and then work out the process that their Microbit will go through to sense the radio message and then create the required response.

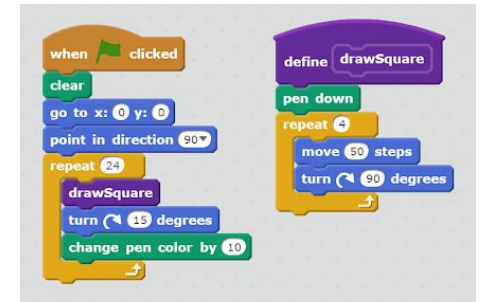
Can find and correct errors in program logic

It is not always guaranteed that learners will come across logic errors, particularly errors where they can learn from resolving the error. Instead, challenge learners to spot errors in pre-written code, either in a shared Scratch program or in code displayed on the board. The Creative Computing guide (<http://scratched.gse.harvard.edu/guide/>) has several debug examples that might be useful for this.



Groups related instructions into named subprograms (in a visual language)

Subprograms or functions can be easily implemented in Scratch by using the Make a Block facility. Learners could make blocks for simple shapes (squares, hexagon, pentagon, circle) and then use the blocks to create spirograph-style art.

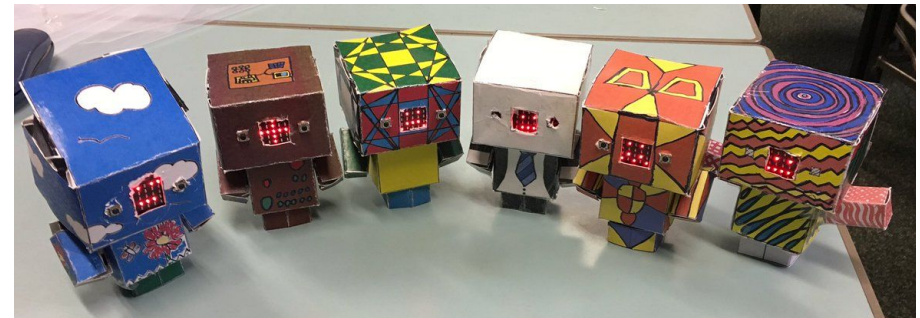


Writes code in which there is communication between parallel processes (in a visual language)

Creating animations in Scratch is a simple way to create parallel processes, with multiple sprites all following their own script when the green flag is clicked. The Broadcast block allows a sprite or stage to communicate commands to other sprites to tell them to start a new section of code. For example, a stage can display an intro screen background for two seconds then communicate to the other sprites by broadcasting a 'Next Scene' message. The other sprites can hide when the green flag is clicked then when they receive a 'Next Scene' message they can show and move around.

Writes code which receives and responds to real world inputs (in a visual language)

Learners could use Microbits to create dice or Rock Paper Scissors games. They could create papercraft characters for a Microbit and then program the device to display different faces depending on whether it is lying down, upside down or being shaken! Microbits could also be programmed to do different things depending on messages from another Microbit device using radio commands.



There are a number of ways to use Scratch with alternative inputs. Teams of learners could work together to create a game and make a game controller using a Makey Makey device or Lego WeDo sensors to control the game. The latest version of Scratch allows learners to easily use a webcam to control sprites using motion detected on the camera.

Designs and builds web pages using appropriate markup languages

Pupils should learn how to create the content of a web page using HTML and change the appearance of their page using CSS. Using a tool such as Codepen (<http://bit.ly/CSScot200>) or Glitch (<http://bit.ly/CSScot201>) allows learners to instantly see the result of their code, and also allows them to publish their work.

Organiser 1 Fourth Level	Understanding the world through computational thinking
<p>Outcome:</p>	<p>Benchmarks:</p>
<p>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</p> <p style="text-align: right;">TCH 4-13a</p> <p>I can informally compare algorithms for correctness and efficiency</p> <p style="text-align: right;">TCH 4-13b</p>	<ul style="list-style-type: none"> 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

What learning and assessment may look like:

Identifies the transfer of information through complex systems involving both computers and physical artefacts

Discuss the importance of being able to track a large number of physical objects within a process, for example tracking people and bags in an airline system going from one country to another or parcels being tracked between a store and an address. Discuss what happens when something goes wrong, such as the customer not being in for delivery or a plane being diverted to a different destination due to a storm.



Learners could plan out or discuss what information and processes would be needed to keep track of people in a Futurama-style tube transport system (<http://bit.ly/CSScot174>), or a pneumatic tube school bag delivery system in their school, with each item needing tagged with an item number and a destination.



Ask pupils to plan a new traffic junction near the school or in the local area. Think about how to make the junction as efficient as possible under different conditions (such as school days, weekends and during fair days or special events). How will the junction know when traffic wants to flow and when pedestrians want to cross, and reduce the amount of waiting time for both of those groups. For example, traffic lights might use motion sensors or sensors in the road detecting the presence of cars, adjusting timing based on traffic volume and flow. Do you allow pedestrians to cross at every cycle or require them to press the button? Do you stop the cars when the button is pressed or do you wait for the next time the cars are stopped anyway?

Ask learners to think about the information and processes that might be involved in growing vegetables and how these these processes could be automated. Show a video of the Farmbot automated systems for gardening (such as <http://bit.ly/CSScot187>). Learners could then identify the steps in the processes required to plant, maintain, and harvest crops and then investigate the data required to allow the each of the stages to be completed by a computer system. Think about how to identify the difference between a weed and a crop, or work out if watering required or if the crops are in good health.



Discuss how self driving cars navigate choosing a efficient and safe route (avoiding obstacles, redirecting based on traffic reports received in real time, choosing a route avoiding roadworks based on crowdsourced information). Students can consider the moral dilemmas in crash situations through discussion and using the quiz at <http://bit.ly/CSScot188> and discuss how their views compare to other learners and to the average on the website.

Describes instances of human decision making as an information process

Discuss the factors influencing how you choose which supermarket checkout queue to pick. This might be number of people in each queue or the number of items they have, or involve more complex conditions such as someone having an item which might take more time to process (someone with a bottle of alcohol who will need to provide ID to prove their age, or an item of clothing with a security seal). Discuss whether their human decisions are based on data or biased factors, such as thinking that elderly people or a parent with a small child will take more time to pay. Learners could then do a homework task where they time how long it takes them to get through the checkout compared to someone in another queue.

Discuss how to split a group of people based on a set of criteria. This might be setting classes in school, choosing where everyone will sit at a wedding dinner or in a classroom, or deciding who to invite to a party or a trip. Learners could think about the criteria they would put in place if they were responsible for setting the class groupings for their year group.

Ask learners to think about how to decide between methods of transport for a journey (this decision making can be computerised to take out the human decision making, such as making a bike-tube barometer to decide whether to cycle to work or take public transport <http://bit.ly/CSScot168>). Discuss what factors influence their decisions in the morning, such as weather, traffic or bus delays, if a family member happens to be driving near school at the right time, or wanting to walk via a friend's house. Can learners then find the most efficient route to school, based on the method of transport? Compare the results of different mapping tools such as Google Maps with real-life results, and log how long it takes to travel each day.

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

Discuss how there are different ways to solve problems such as loading a dishwasher, planning a journey, or strategies for playing Monopoly and other board games.

Look at algorithms for finding a value in a list (for example binary search vs linear search). Learners could race to find an item (such as a name in a phone book or a word in a dictionary) and compare the speed of different approaches. Learners could do a similar exercise with sorting algorithms, racing to sort Top Trumps cards, library books, or even various objects in weight order.

Discuss two or three different sorting algorithms (such as quicksort, bubble sort and insertion sort) and compare the number of comparisons, number of swaps, memory requirements, and overall time taken (<http://bit.ly/CSScot184>). Learners can consider situations where one factor might be so important that other factors don't matter. For example, when sorting library books with a small amount of desk space, or sorting people into height when you can only compare one person at a time. There are a number of unplugged activities for exploring sorting algorithms (<http://bit.ly/CSScot159>) and sorting networks (<http://bit.ly/CSScot169> and <http://bit.ly/CSScot158>).

Organiser 2 Fourth Level	Understanding and analysing computing technology
Outcome:	Benchmarks:
<p>I understand constructs and data structures in a textual programming language TCH 4-14a</p> <p>I can explain the overall operation and architecture of a digitally created solution TCH 4-14b</p> <p>I understand the relationship between high level language and the operation of computer TCH 4-1c</p>	<ul style="list-style-type: none"> • 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 between different levels of abstraction
What learning and assessment may look like:	
<p>Demonstrates an understanding of how visual instructions and textual instructions for the same construct are related</p> <p>Learners could use a language that allows users to switch between blocks and text, such as Blockly, PencilCode or the Microbit Javascript Blocks editor. This will allow users to compare simple block programs and it's text equivalent. They should try creating a block program and then making changes to the text and see the effect this has on the output. Having been given example programs in both types of language, learners could match visual programs and textual programs that would produce the same output. This will also reinforce the idea that there are multiple ways to create a solution to a problem.</p> <p>Learners could create a dictionary matching visual language block commands to their textual language equivalent. Ideally this should also include a definition of each command, written by the learner. This dictionary can be started with a basic set of commands and then developed gradually by the learner as they come across new commands. They could then use this dictionary when creating programs in Organiser 3.</p> <p>Understands basic control constructs such as sequence, selection, repetition, variables and numerical calculations in a textual language</p> <p>Learners can read the commands in a textual language program (of similar complexity to a Third level program in a visual language) to predict what the program will do. They can pick out and describe the precise meaning of individual structures within a larger textual language program.</p> <p>Learners could take jumbled lines of a short program and try to rearrange them into order. This type of problems are called Parson's Problems (http://bit.ly/CSScot170). Learners can discuss logic error that might occur if the lines of code are in the wrong order.</p>	

Identifies and explains syntax errors in a program written in a textual language

They can identify when there are mistakes in a program written in a textual language (and understand the difference between logic and syntax errors). Learners could be given example programs containing syntax errors in order to identify and explain the syntax errors present. This can be done on paper if the code is given along with compiler output, to help learners identify the location of the bugs. Learners could take a pre-written program and add in different types of errors for a partner to spot.

Demonstrates an understanding of representations of data structures in a textual language

Learners should be about to create and manipulate strings and arrays in a textual language. Learners could manipulate text in fun ways, such as using substring commands in order to select parts of strings such as generating their 'Star Wars name' (which consists of the first three letters of your first name, first two letters of your surname, three letters from your mother's maiden name and two letters from your town of birth. This is also a great opportunity to discuss phishing with learners!) Learners could use dice to randomly choose values from arrays in order to create a nonsense sentence or haiku.

More advanced learners may start to move into an understanding of objects, attributes, and methods if they are using a object oriented textual language such as Greenfoot for Organiser 2 and 3.

Demonstrates an understanding of how computers represent and manipulate information in a range of formats

Learners could explore how images and text can be stored and can be manipulated by computers. There are excellent online resources available on Multimedia Computing that allow learners to easily manipulate images and data while learning Python commands (Learners' guide <http://bit.ly/CSScot172> and teachers slides <http://bit.ly/CSScot173>).

Before learners can create relational databases, they will first need to learn about the concepts involved (such as Organiser 1) and then find out about the tools they can use (such as Organiser 2). Learners could learn about the flaws in flat file databases, perhaps identifying data anomalies in a database (such as insertion, deletion, modification issues). Learners are then introduced to the concept of splitting databases into multiple tables and linking them together, and a need for a unique identifier. They will learn how to use the database tools they need before they can create and use a database.

Demonstrates an understanding of program plans expressed in accepted design representations, for example, pseudocode, storyboarding, structure diagram, data flow diagram, flow chart

Learners should be able to identify familiar control constructs within a program created using an accepted design notation. They should be able to identify logical errors contained within a program plan. This will help them understand that planning can allow logic errors in complex algorithms to be spotted before time is wasted translating into a program.

Explain to learners that knowledge of design notations allow programmers to discuss algorithms and program design without needing to know a shared language syntax. This is particularly important when those programmers know different languages from each other.

Learners could be given the parts of a program expressed in a design notation and place them in the correct order to complete a task, or to identify intentionally missing constructs. For example, a task that needs a fixed loop but this wasn't provided.

Demonstrates an understanding of the underlying technical concepts of some specific facets of modern complex technologies

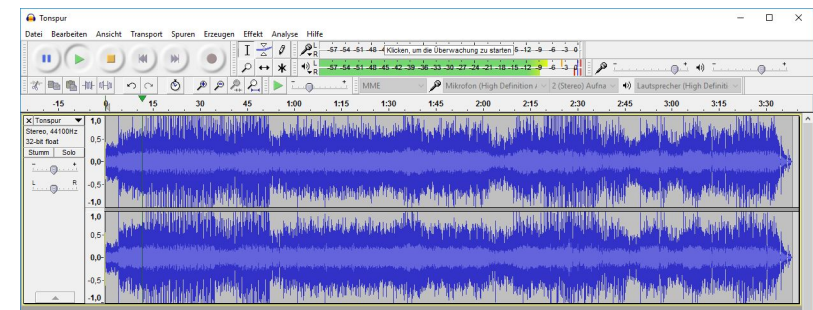
Learners could work in groups to research a technology that they have chosen from a list. They can find out how it was developed, why it was originally created, and the basics of how it works. They could then produce a poster, give a presentation, or make a video, animation or webpage on their chosen technology. Technologies could include online payment systems or mobile payments, SATNAV or GPS mapping, self-driving cars, VR headsets, smart watches, games consoles and hardware, online multiplayer gaming, 3D printers, mobile phones, internet communications, and flexible displays.

Demonstrates an understanding that computers translate information between different levels of abstraction

Functions in programming languages can be used without knowing how that function works just what goes in and what comes out. You don't need to know how addition will be performed by the processor in order to use the addition command. However, it is important for learners to have an understanding of how computers work and what the different levels of software are doing when responding to a command from a user. It is important that they understand that the computer is not magic!

For example, when a computer does something unexpected (such as moving a photo around a Word document when another photo is resized), learners could try and work out why that has happened, why the software has responded that way to a command (like finding out that photos are anchored to a particular point in a Word document and will move about if the anchor moves).

Learners should also understand that information can be represented in different ways within a computer. For example a computer will save a sound recording by digitise an analogue sound from a microphone. Even though it may be represented as an analogue waveform in software such as Audacity, it is still saved as a digital file, electronically as 0s and 1s. It is the same for computer programs that learners may create. The programs they write, using block commands or typed words, as then translated by the compiler software into machine code in order for the computer to understand it and implement the instructions.



Organiser 3 Fourth Level	Designing, building and testing computing solutions
Outcome:	Benchmarks:
<p>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.</p> <p style="text-align: right;">TCH 4-15a</p>	<ul style="list-style-type: none"> • 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.
What learning and assessment may look like:	
<p>Analyses problem specifications across a range of contexts, identifying key requirements</p> <p>Learners can identify input, process and output from a task specification. They can identify the data, where it will come from, and the operations that will be performed using that data.</p> <p>Fourth Level learners can still use visual languages (although ideally in an environment that allows a visual to textual transition), but doing more challenging activities than Third Level learners. They will be independently be analysing the specification and planning their response to the design challenge. For example, a Fourth Level learner might create a more complex game utilising a physical device as an input, such as designing and creating a game with a Makey Makey used as the basis for a custom-designed controller.</p> <p>Writes a program in a textual language which uses variables and constructs such as sequence, selection and repetition</p> <p>At Fourth Level, learners should be starting to learn how to code in a textual language, ideally starting with an environment that allows users to switch between blocks and text commands. Using an environment such as Blockly, PencilCode or the Microbit Javascript Blocks editor will allow Third and Fourth Level learners to complete the same task but in different types of environments.</p> <p>Learners must have an opportunity to learn about the textual language, investigating the commands they can use (as they would in Organiser 2) rather than just looking at a piece of code and trying to modify one simple thing such as changing a variable name or a calculation. They should be thinking about what each line of code is doing before they adapt code. For example, they could be shown code for making a sprite turn left, and adapt that code to turn right, up and down. They must be given the opportunity to read and adapt simple code, or scaffold their learning with example code, before they attempt to write their own programs. Learners could</p>	

also create a language dictionary, where they list the terms in a textual language, identify the command in a block based language that does the same thing, and describe the effect or purpose of that command.

Creates a design using accepted design methodologies

Learners will be able to analyse a problem and design a solution using a technique such as pseudocode, storyboarding, structure diagrams, data flow diagrams and flowcharts. Storyboarding might be a particularly accessible technique that is suitable for both Third and Fourth Level learners when planning out an animation in a visual language. Flow charts might then be a good next step, with learners planning what happens in a textual program depending on input from a user.

Develops a relational database to represent structured information

Learners will be able to create multiple tables within a relational database application and implement relationships between these tables. Using this database, learners will be able to perform search and sort operations. Learners can create reports selecting the fields from different tables to display only the field required.

Learners can be given the data to be imported into their database or could design a database to answer a question about an open dataset they have found themselves. Providing a large data set to be imported will give learners a better understanding on the power and benefit of databases.

Debugs code and can distinguish between the nature of identified errors e.g. syntax and logic

Learners will be encountering syntax errors for the first time, as these are usually not possible in a visual language. They will need to learn strategies for debugging these errors, such as having a checklist for spotting the most common mistakes in a particular language (such as ensuring each line ends in a ';'). It would be good for learners to be given buggy programs where they can try to spot all the errors.

Writes test and evaluation reports

Learners do not need to create test reports for their own code. Learners could be asked to design tests to a problem requirement specification and be given programs to solve those problems. They could test and identify solutions that work perfectly, don't work at all, or mostly work but have a flaw somewhere (however, they are not necessarily given the code to find the error and fix it).

Can make use of logical operators in a textual language

Learners should learn how to use AND, OR, NOT conditions in a textual language within conditional loops and selection statements. They will have learned about the concept of logic at First Level in Organiser 1, and should have implemented it in a visual language at Second Level in Organisers 2 and 3 through creating games.

If learners are creating games in a textual language then the context for such statements will be similar to visual languages. For example:

- if left key pressed AND NOT touching wall then move left
- repeat until score is more than 10 OR character is touching water,

- loop the racing car instructions while remaining time is greater than one AND the car is not touching the red finish line.

If learners are solving real world problems in a textual language, then it could be

- repeat asking for username and password until username equals MrAwesome AND password equals 123456789
- if time is before 1530 AND temperature is less than 22 then turn the heating on, else do nothing.

Learners should also be able to make use of comparators, such as '=', '<', '>' in their programs.

Writes a program in a textual language which uses variables within instructions instead of specific values where appropriate

Learners could write a program using pythagoras theorem to calculate the hypotenuse length in a triangle, where the user enters the other two side lengths. They could create programs for other appropriate maths formulae.

Learners could use variables to control the speed of characters within a game created in a textual language. They could then discuss the advantages of using variables rather than numbers (for example, it is easier to change the function of the program using the variable rather than numbers in every statement).

The result of a calculation could be used in a subsequent selection statement, for example if profit is more than last profit display the text “awesome”, else display “work harder”

Designs appropriate data structures to represent information in a textual language

Once learners understand the concept of arrays (such as in Organiser 1) and has learned commands and blocks of code for using arrays in a textual language (such as in Organiser 2) then they can use them in their own designs and programs. Third Level learners in a class could use a list function in a visual language to keep track of players names and high scores while Fourth Level learners attempt this in a textual language.

Selects an appropriate platform on which to develop a physical and/or software solution from a requirements specification

By Fourth level, learners should hopefully have encountered a number of different development environments, possibly including physical computing environments such as Microbit development environments. They should be given the opportunity to select the most suitable or preferred environment to use to produce a solution to a challenge.

Compares common algorithms for example, those for sorting and searching, and justify which would be most appropriate for a given problem

Learners could be given prewritten implementations of different algorithms within a textual programming environment that they could call within a solutions they have developed, or even block-based code for a physical system such as Microbit (<http://bit.ly/CSScot185>). They could observe the performance of each algorithm in their program, then, with their knowledge from Organiser 1, make a recommendation based on relative speed and resources required.

Learners could then use the results from this Organiser 3 task to expand their knowledge at Organiser 2 level. Learners could be given specifications for a number of hardware systems that would run programs using sort or search algorithms. They could make recommendations based on their knowledge of speed, process, complexity, and storage requirements to rule out unsuitable algorithms for a particular system. For example, an algorithm (such as a selection sort) might require a lot of RAM and be unsuitable for a hardware system with limited RAM. A system scanning people's tickets into a sports stadium needs to perform a fast search.

Designs and builds web pages which include interactivity

Learners should be able to add simple JavaScript to a webpage to allow interactivity. This will be simple code mostly focused around using the mouse to interact with elements on the page. For example, learners could use Onmouseover and Onmouseout JavaScript attributes within an HTML tag to allow the colour to change when the mouse is moved over and then away from a piece of text. More advanced learners (who have maybe learned some JavaScript as a textual language) could use functions to display alerts when a button is pressed, to make changes to the content of the page, or create a simple calculator page in conjunction with input tags.

Learners can continue to use a tool such as Codepen (<http://bit.ly/CSScot200>) or Glitch (<http://bit.ly/CSScot201>) to instantly see the result of their code. This also allows them to publish their work online with their JavaScript enhanced pages.

Planning Progression Pathways

Planning Progression across the three Computing Science Organisers

It is important to not view the benchmarks individually and in isolation. Schools should instead plan out how learners will progress through experiencing groups of benchmarks that cover outcomes in the three organisers in different contexts each year. The Organisers for Computing Science are a spiral curriculum. It is expected that learners will approach themes and contexts multiple times throughout their Broad General Education, building up and expanding their understanding each time. A spiral-based curriculum for Computing Science as part of their core entitlement for Broad General Education will support learners develop their skills further each year to prepare them for suitable qualification pathways in Computing Science. This section will explore examples of this in various CS contexts, leading to different progression pathways.

As they progress each year through their Third and Fourth levels, learners may also revisit outcomes and benchmarks several times through different contexts, allowing them the opportunity to secure their knowledge of skills and concepts by applying it in different ways. It is recommended that schools plan for the contexts that learners will encounter if they choose to continue studying for a qualification in Computing Science.

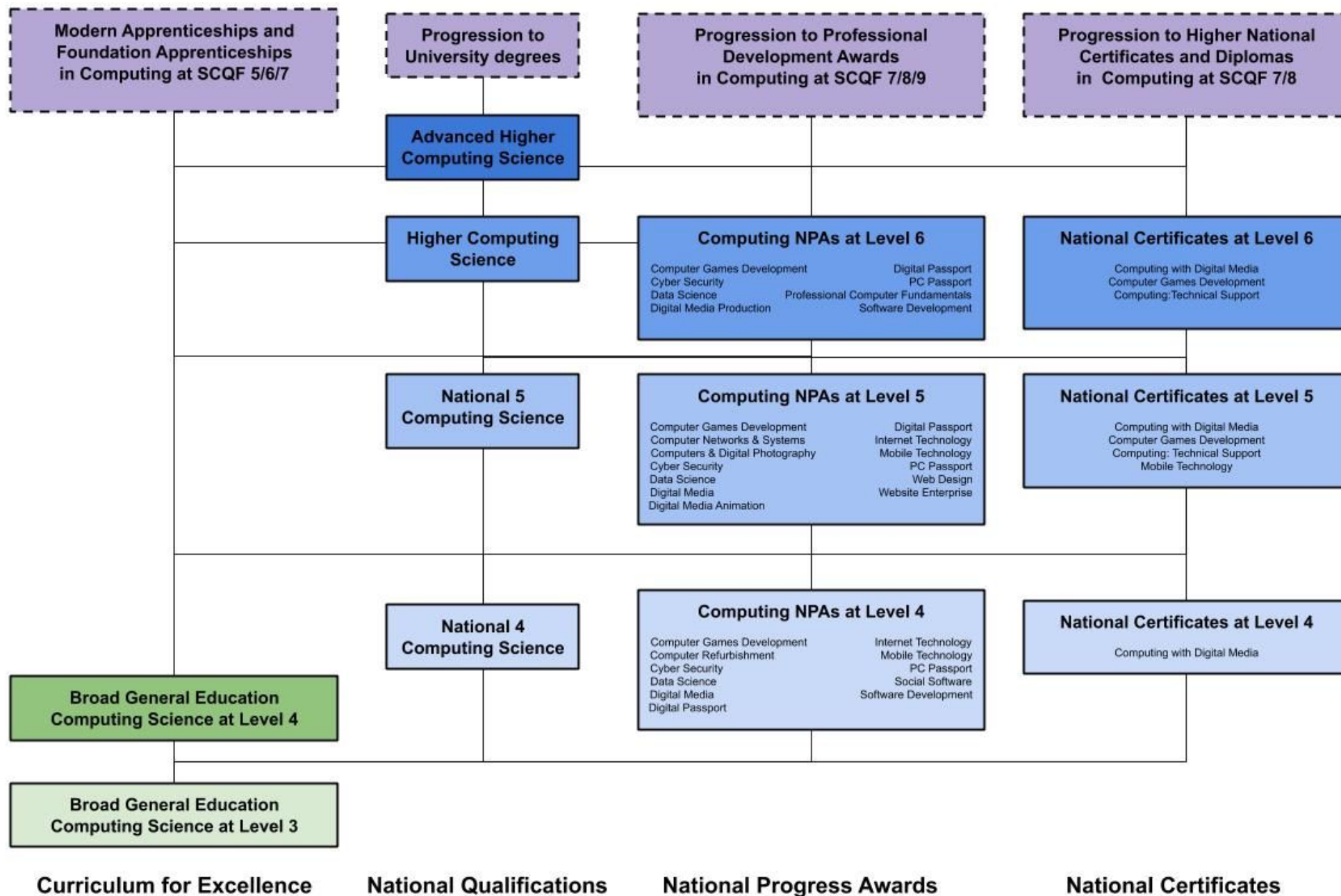
Possible Qualification Pathways

The most common CS qualification for learners following Broad General Education is to move on exam-based National qualifications, with some learners electing to continue to Higher and Advanced Higher Computing Science. However, other National Qualification Group Awards are possible pathways, particularly National Progress Awards (NPAs). These qualifications are aimed at assessing a defined set of skills and knowledge in specialist vocational areas. They also link to National Occupational Standards. They consist of a number of units (usually around three or four) and are internally assessed through coursework and assessments. There is a wide range of Computing NPAs available to schools (<http://bit.ly/CSScot183>), including Cyber Security, Computer Games Design, Digital Media Computing, Animation, Web Design, Software Development, and Data Science.

Tailoring BGE provision to suit qualification pathways

Once schools have identified suitable pathways for learners in Computing Science, they should then ensure that learners' experiences in their Broad General Education entitlement will support them to develop the most appropriate skills to prepare them for those pathways. For example, if a school is offering a pathway option of studying for an NPA in Data Science, then learners would ideally have encountered and used a data analytics tool as a context for learning about structured information and data storage in BGE, whereas a school offering National 4 and 5 may prefer to focus on storing and manipulating information in databases.

It is common for CS educators to focus on the end product and assessment task when planning units of work, such as whether the learner will be creating a database, a computer game, or an animation for example. If this is the starting point for planning, then it is suggested that after choosing an assessment method or practical activity, educators should think approaches for teaching the tools and languages that learners will need to carry out that task (Organiser 2) and the concepts that learners will need to understand (Organiser 1). It is also useful to think about the possible misunderstandings that learners may acquire, and address these when teaching concepts and tools.



Web Development: Progression Pathway

Teachers should use the benchmarks to help make assessment judgements on the organisers in different contexts. For example, a school might choose to use web development as a context for learning to prepare learners for the web development content in National 5 Computing Science or allow them to progress on to the National Progress Award on Web Design at Level 5.

Exploring a web development context within the spiral nature of the CS curriculum, learners might learn about markup languages and create simple HTML web pages in S1, then in S2 they will revisit the concepts of HTML tags but also learn about style sheets before creating web pages using both HTML and CSS. In S3 they may add simple interactivity with lines of JavaScript.

Web Development: Planning using the Benchmarks and Organisers

The first spiral round web development would likely involve learning about how to represent and manipulate structure information using HTML code (Organiser 3). Prior to this they will need an understanding of structured information in web pages, and how web browsers display the code that is loaded (Organiser 2). However, first they will need to understand that we can identify different parts of a page with similar characteristics such as headings, paragraph text or bullet points.

When revisiting web development next, learners could find out how to change the visual style of the content of a webpage by using Cascading Style Sheets (Organiser 3). In order to be able to program using CSS code, learners will first have to understand the separation of content and appearance, and that this allows us to represent the same information in different ways by changing the CSS code, or we can style different pages the same by using the same style sheet (Organiser 2). They will first need an understanding that although we can change the appearance of all of the text on a page, we can also uniquely identify a particular word or paragraph and give it different characteristics (Organiser 1).

At Fourth level, learners would be revisiting the web development context by learning about how to add basic interactivity to web pages using simple Javascript code (Organiser 3). To do this, they will first need to have an understanding how information transfers from web servers to be displayed on different physical devices such as tablets or mobile phones (Organiser 1) and then find out a little about specific facets of the underlying technical concepts of client side scripts differing from server side scripts (Organiser 2).

In this way, learners build up their understanding each time revisiting the context. They need to work through the Organisers in order to understand each new concept, but spiralling around a context develops their skills and understanding further.

Computer Games: Progression Pathway

Computer games development skills can be taught in a simple but engaging way in BGE as a pathway to the NPA in Computer Games Development, perhaps leading learners to study the NC in Further Education or go on to one of the many degree programmes available.

Schools wanting to prepare learners for a pathway involving NPAs in Computer Games Development may choose to introduce learners to increasingly complex games development environments and concepts each year as a context for learning about benchmarks in programming and arrays. They could start with simple games in Scratch in S1, scrolling games in S2 then introducing learners to creating games in Greenfoot in S3 or getting learners to work in teams to develop a physical computing interface for their games using Makey Makeys.

Computer Games: Planning using the Benchmarks and Organisers

In BGE, learners might be given the task to create a computer game using a visual language (Organiser 3) that uses more than one variable. For example, a game that stores a player's score, lives remaining, and time survived. For a more secure learner, a scrolling game might be a suitable challenge, with variables for storing sprite positions, movement speed, gravity and time taken.

In order to achieve this Organiser 3 activity, first think about all the skills learners will need to learn in order to create a game. These skills can be taught and explored as part of Organiser 2, where learners can be introduced to individual blocks and example programs to help them gain the skills they will need to independently create a game. For example, if the learner will be creating a game with moving sprites, then they will need to learn different ways of having that sprite move (such as following a mouse, chasing another sprite or controlled by arrow keys).

Different strategies could be used to explore programs and commands, such as predicting the output, or seeing the output and guessing what blocks are used, or getting jumbled lines of code and putting them into the correct order. Learners could be given sections of code in which conditional statements involve the comparison of multiple variables and asked to state the values of variables at different points during the code.

However, in order to develop these skills in Organiser 2, we must first think about the concepts the learners will require, which should be taught as Organiser 1. For example, learners could be introduced to storing multiple pieces of data and using these to make decisions. This could be through tracking values as the class are read a short history about a computer games company (numbers of games sold, companies value, number of different games mentioned, etc). Learners could create instructions to guide each other around the classroom to find objects in a particular order, in a similar manner to how they will instruct their sprites in Organiser 3. This could even be a whole class activity, creating a real life computer game with different learners creating and following instructions as sprites or tracking different variables.

Animation: Progression Pathway

Computer animation skills can be taught within the BGE using a visual programming language (instead of or in addition to a variety of computer-based stop-motion animation techniques and apps). This can lead to the NPA in Digital Media Animation or the NPA and NC awards in Digital Media Computing.

Learners may learn how to make a simple scene initially, expanding their skills later by having multiple scenes and sprite processes controlled by 'broadcast' blocks. Learners could introduce more complex audio and still image editing skills by developing digital media techniques and combining them in an animation. Exported animation files could then even be embedded in a larger video showcase.

Animation: Planning using the Benchmarks and Organisers

Learners might be given a task to create an animation in a visual language such as Scratch (Organiser 3) that uses a broadcast block to make multiple actions happen simultaneously. For example, they could have the background will change and two characters will move and start a conversation or tell a joke.

In order to achieve this Organiser 3 activity, learners will need to learn about specific command blocks (Organiser 2). Learners could be introduced to the broadcast block through demonstration. They could be given sections of code to make predictions of what that code will do using the newly introduced blocks. Students will explore the difference between broadcast block and the wait block using an example program to experiment with the two blocks. They could discuss the advantages of the two approaches.

However, before being introduced to the blocks within Scratch students could be shown the concepts of parallel processes, this could take the form of reading a script and following stage directions or building lego models in groups where each person has a different part of the instructions (some of which can be done at the same time and others that require a prior step).

Data education: Progression Pathway

Key concepts in data literacy and data education can be taught in the BGE to prepare learners to study the NPA in Data Science, to help them understand what is involved in this field and show it as an attractive subject choice with relevance across the curriculum. Skills in this area, such as being able to get meaning from data visualisations and to analyse tables of data to spot patterns and see trends.

Data education: Planning using the Benchmarks and Organisers

Learners can explore different ways to represent and store data in programs, databases and spreadsheets (Organiser 3). However, they need an understanding of how structured information works in these environments and applications (Organiser 2). First of all though, they need to understand that we can have a set of characteristics about a related set of items, and that items can have attributes that we use to describe it (Organiser 1). For example, learners might explore structured information using a sticky note database or Top Trumps game and then create their own game in a database, finding out about fields and records.

If learners are to use a visual language to store and manipulate information, they should first learn how structures such as lists or arrays can be used to store multiple values for a single piece of information (Organiser 1). They could find out the benefits of lists compared to single variables. Then they should learn how to create and use these structures in a visual language (Organiser 2). For example, learners could be shown how to access values in a list. Finally, learners will make use of list structures within their programs (Organiser 3). For example, they might add a player's name and score to lists to create a high score table in a game.

Support, Resources and Competitions

There is a huge range of activities and resources that are suitable for teaching the three Curriculum Organisers at Third and Fourth Levels. However, there are some websites that we feel are particularly useful for teaching across the levels, or feature a large collection of suitable 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 Glow community <http://bit.ly/CSScot186>

Collected resources to support learning and teaching of Computing Science.

CompEdNet: <http://bit.ly/CSScot198>

The Scottish Computing Science teachers' forum to discuss and share ideas and resources.

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

Raspberry Pi Foundation: <http://bit.ly/CSScot192> and <http://bit.ly/CSScot191>

The Raspberry Pi education site features links to useful resources for educators. They also have a section that includes all their projects for learners, including resources for Scratch, Python and web development.

Microbit Foundation: <http://bit.ly/CSScot193>

The Microbit website has a variety of ideas and teaching resources. If you are interested in teaching Computing Science through the use of the Microbit, then CS4all and the NYC Department of Education have a set of resources for teaching using the Microbit. <http://bit.ly/CSScot197>

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.

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 Secondary pupils. They involve fun activities, puzzles and a bit of magic!

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.

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 run by the University of Manchester. The competition deadline is usually the end of March each year.

Alan Turing Cryptography competition: <http://bit.ly/CSScot194>

An annual competition for teams of UK learners up to S4 run by the University of Manchester. The competition usually runs from the end of January to early April each year. New chapters are released every week or two during the competition, each with a code to crack.

National Cipher Challenge: <http://bit.ly/CSScot195>

An annual codebreaking challenge for Secondary learners in the UK run by the University of Southampton. The challenge usually runs from October to December each year. The competition is structured as a series of encrypted messages which tell a story. There are teaching resources too.

BAFTA Young Games Designer Competition: <http://bit.ly/CSScot180>

An annual computer games design and making competition for learners aged 10 to 18. The competition deadline is usually the end of March each year.

Apps for Good: <http://bit.ly/CSScot190>

Apps for Good offer several courses and mini courses that can be run in schools, including app design, app development, machine learning and Internet of Things. They run annual Awards to celebrate work created by learners, with a closing date towards the end of April each year.

CyberFirst: <http://bit.ly/CSScot196>

The National Cyber Security Centre runs courses on cybersecurity, an annual CyberFirst Girls competition, and an online course called Cyber Discovery. The CyberFirst Girls competition registration begins in December each year, with the first round taking place in January.

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) (<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) (<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) (<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 programming 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) (<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 *outputs* 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) (<http://bit.ly/CSScot148>), [inputs](http://bit.ly/CSScot141) (<http://bit.ly/CSScot141>) and [outputs](http://bit.ly/CSScot142) (<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) (<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) (<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) (<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) (<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 programming 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 π 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) (<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.
- **Curriculum Organisers** 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

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/

Scratch cards in action by Bobby Lewis [@usabbs](https://twitter.com/usabbs/status/746708479554035712), <https://twitter.com/usabbs/status/746708479554035712>

Microbit rock paper scissors game by Microbit Foundation <https://makecode.microbit.org/projects/rock-paper-scissors>

Origami instructions https://commons.wikimedia.org/wiki/File:Origami_windmill_base.svg

Audacity screenshot https://commons.wikimedia.org/wiki/File:Audacity_Version_2.1.2.PNG

Traffic junction by Sandid on Pixabay <https://pixabay.com/photos/cross-road-junction-road-traffic-425054/>

Pedestrian crossing by kyrien at iStockphoto <https://www.istockphoto.com/gb/photo/pedestrian-button-indicating-wait-in-london-uk-gm898602050-247966017>

Farmbot <https://en.wikipedia.org/wiki/FarmBot>

Caesar Cipher wheel https://en.wikipedia.org/wiki/Caesar_cipher

Web colors table https://en.wikipedia.org/wiki/Web_colors

Futurama image copyright owned by Matt Groening and 20th Century Fox Television.

Scratch spirograph blocks and photo by authors

Microbit papercraft photo by authors

Microbit invaders game photo by authors



This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-nc-sa/4.0/).