A progression of programming concepts and skills from Foundation to Year 6

Computing- Coding- Key Concepts, Skills and Approaches to Programming

These are the over-riding themes, concepts and approaches that under-pin all programming.

COMPUTATIONAL THINKING

Computational thinking is about looking at a problem in a way in which a computer can help us to solve it. It is defined as the process of formulating and solving problems by breaking them down into simple steps. This is a two-step process:

First, we think about the steps needed to solve a problem. Then, we use → our technical skills to get the computer working on the problem.

For a computer animation, for example, you'll first plan the story and how it will be shot. Then, you'll use the computer hardware and software to create the animation.

Computational thinking is not thinking about computers or like computers. https://youtu.be/91utNt5qshE

DECOMPOSITION
Breaking down into parts

In computing, decomposition is the process of breaking down a task into smaller, more-manageable parts. It has many advantages. It helps us manage large projects and makes the process of solving a complex problem less daunting and much easier to take on.

PATTERN SPOTTING
Spotting and using similarities

Patterns are everywhere, for example, we use weather patterns to create weather forecasts.
By identifying patterns we can make predictions, create rules and solve more general problems.
Children need to be able to identify repeating patterns in a list of commands to understand how this could be made more efficient using a repeat loop.

ABSTRACTION
Choosing what's important

Abstraction is about simplifying things – identifying what's important without worrying too much about detail.
A school timetable is an abstraction of what happens in a typical week. It shows key information about classes, teachers, rooms and times but ignores further layers of detail such as learning objectives and activities.

ALGORITHMS
How to get it done

An algorithm is a sequence of instructions or a set of rules to get something done.
You'll favour a particular route home from school – you can think of it as an algorithm. There are plenty of alternative routes home, and there'll be an algorithm to describe each one of those too. There are even algorithms for deciding the shortest or fastest route, such as form the basis of satnav systems.
Algorithms are written for a human, rather than for a computer to understand. In this way, algorithms differ from programs.
The main difference is between the two is that an algorithm is a step-by-step procedure for solving the problem while programming is a set of instructions for a computer to follow to perform a task. A program could also be an implementation of code to instruct a computer on how to execute an algorithm.

DEBUGGING
Finding and fixing errors

Errors in algorithms and code are called 'bugs', and the process of finding and fixing these is called 'debugging'. Getting pupils to take responsibility for thinking through their algorithms and code, to identify and fix errors is an important part of learning to think and work like a programmer:

1. Predict what should happen.
2. Test -find out -exactly what happens when a program is run
3. Work out where something has gone wrong.
4. Fix it.

LOGICAL REASONING
Predicting and analysing

If you set up two computers in the same way, give them the same instructions (the program) and the same input, you can pretty much guarantee the same output. This means that they are predictable. Because of this we can use logical reasoning to work out why something happens. Part of logical reasoning is the ability to use existing knowledge to make reliable predictions about future behaviour of a system.

EVALUATING
Making judgements

Evaluation is about making judgements, in an objective and systematic way where possible.
Children need to evaluate the effectiveness of their programs in solving a specific task. Pupils should be encouraged to reflect on their programs- do they do what they wanted? Is there a better way or improvements that can be made?

TINKERING
Exploring and applying learning

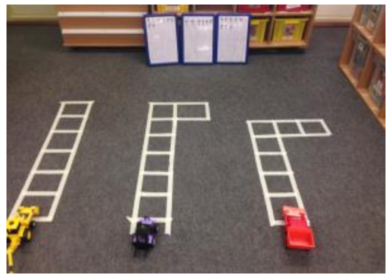
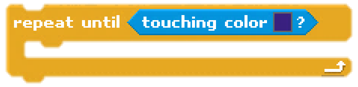
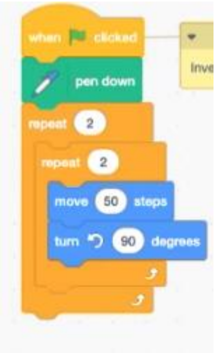
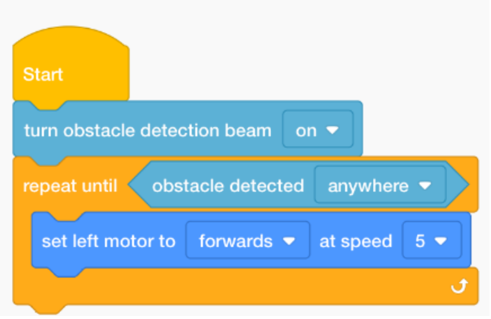
We often try out something new to discover what it does and how it works: this is tinkering. It's closely associated with logical reasoning. Pupils build up experiences of cause and effect: "If I move this, then this happens." It's a big part of independent learning, without your lead. For young children, it's the vital play-based experimentation phase, full of questions and surprises. Ideas which seem wrong can be tried, just to see what happens.

As pupil progress through school, tinkering is more-purposeful exploration and making, often through trial and improvement. It helps us to see our use of technology as being about developing our own understanding, rather than getting a 'right' answer; we may be able to do things in many ways. When using technology which we've tinkered with, we're more likely to be open to novel and innovative solutions.

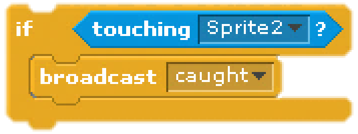
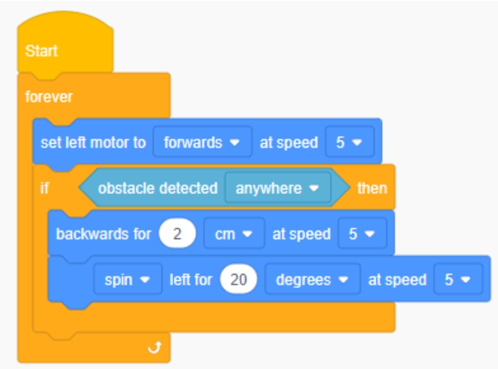
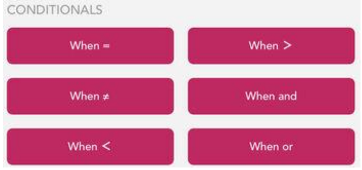
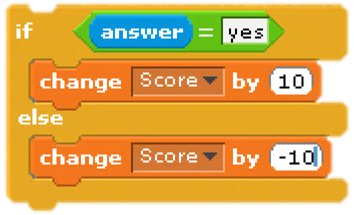
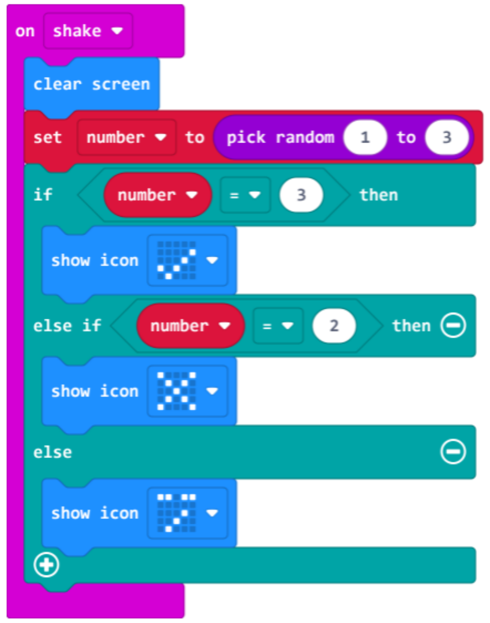
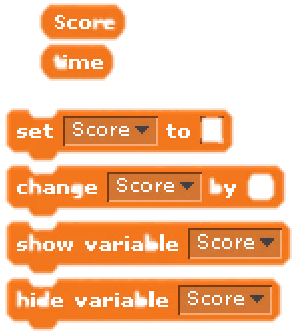
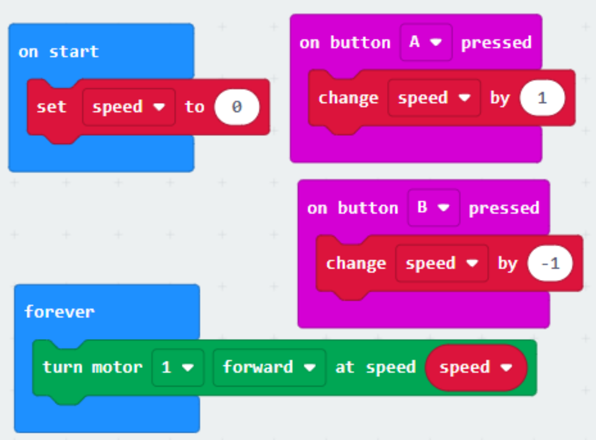
Coding language/ app progression

	Nursery	Reception	Year 1	Year 2	Year 3	Year 4	Year 5	Year 6
On screen code			Beebot app	SCRATCH JR	CODE FOR LIFE	Hopscotch SCRATCH	SCRATCH	Micro:Bits • Applied to DT fairgrounds model • Additional of programmable lights Edison robots Use of sensors for robot to react independently.
Physical/ applied coding		Beebots	Beebots			Crumble computers	Micro:Bits	

A progression of programming concepts and skills from Foundation to Year 6

	FS	Year 1	Year 2	Year 3	Year 4	Year 5	Year 6
SEQUENCING	Sequence forwards and turns e.g. with Beebot **Predict** the outcome of a set of instructions and test the results. EG: 'What shape would this make?' Use symbols to represent an instruction e.g. ↑→ for forward and turn. Know how to clear the code **Decompose** by breaking the code down into chunks (1 step at a time) 1) (clear) 2) (clear) 3) (clear)	Follow a given sequence including forwards, turns and backwards. Know that the order of instructions is important. Write a **sequence** for others to follow. **Decompose** by breaking the sequence into chunks. **Predict** the outcome of a set of instructions and test the results.  Use symbols to represent an instruction in the correct order. e.g. ↑→ for forward and turn. Know how to clear the code	**Sequence** instructions including forwards, back and turns more efficiently. Understand that a sequence of instructions needs to be clear, precise and unambiguous. Understand that the order in which instructions are given will make a difference to the outcome. Understand that the direction and amount of turn is relative to the position of object – on screen or in real life – that is being moved.	Sequence instructions in the correct order with increasing number of commands. Understand that a sequence of instructions is called an **Algorithm** and that the instructions for a computer to follow is a **program**. Amount of turn is given as a number of quarter turns, not in number of degrees.	Sequence instructions in the correct order to create an animation sequence, draw a shape or solve a problem. Understand that a sequence of instructions is called an **Algorithm** and that the instructions for a computer to follow is a **program**. Amount of turn in an program to be given as a number of degrees. Be able to assess success of given instructions and identify and correct any errors that occur. Be able to evaluate the effectiveness of an algorithm written by their peers in class.	Understand and use algorithms which include: • Repeat loops • Event handling • Selection	Understand and use algorithms which include: • Repeat loops • Event handling • Selection • Variables Understand and use interrupts when certain events occur.
REPEAT LOOPS			Understand how to read and interpret a repeat in loop in an algorithm (set of instructions) Use a number to specify movement rather than repeated commands (e.g. in Scratch Junior forward 4 rather than ↑↑↑↑	Understand informal notation for showing a move is repeated. E.G [→] x 3 = move right 3 times	Understand what simple **loops** and **repeats** are and how they can make a program more efficient. Use count controlled repeat loops **Pattern spotting** - be able to identify which commands need to be repeated and how many times to achieve a desired end.	Use the instruction **repeat until …**  repeat until touching color ■ ? *Example code from Scratch3* Read, write and debug **nested loops** (loops within a loop) e.g. creating an algorithm to draw a square, then put this algorithm inside another loop to create a repeated pattern.  when 🏳 clicked pen down repeat 2 repeat 2 move 50 steps turn ↺ 90 degrees	Use a variable and **operators** (the green blocks in Scratch) within a loop to govern termination:  Start turn obstacle detection beam on repeat until obstacle detected anywhere set left motor to forwards at speed 5 *Example code from Edscratch for Edison*

A progression of programming concepts and skills from Foundation to Year 6

Selection (event handling)	Know that when I press GO the sequence will run. Use different action bricks in the Duplo train to make things happen	Know that when a key (e.g. space bar) is pressed, the sprite/character will move.	Control a character in a game or animation where clicking make something happen.	Be able to create an animation or game where clicking on certain 'triggers' (objects/sprites/keys) will cause something to happen.	Be able to use a range of **inputs** to start an event or control a character e.g. space bar, mouse click, ipad press. **Threads (parallel execution)** – Allow more than one event to happen at the same time e.g. having more than one set of blocks or instructions running at the same time.	In Scratch use a **broadcast** to co-ordinate events in a program with more than one sprite(one event causes another to happen eg. Game over)  if touching Sprite2 ? broadcast caught *Example code from Scratch3*	Use events to interrupt program and run sub routines Edison:  Start forever set left motor to forwards at speed 5 if obstacle detected anywhere then backwards for 2 cm at speed 5 spin left for 20 degrees at speed 5 *Example code from Edscratch for Edison*
Conditional Statements					Understand that we can make actions occur only under certain conditions.  CONDITIONALS When = When > When ≠ When and When < When or On Hopscotch, there are 'Whens' for events that happen on your iPad eg WHEN the iPad is shaken, play a pop sound. WHEN the up arrow is pressed, make the character jump.	Use 'if, then, else' statements e.g. in a quiz: if answer correct…  if answer = yes change Score by 10 else change Score by -10 *Example code from Scratch3*	Use selection to govern different events using the 'if / else' Eg. Microbit 8 ball  on shake clear screen set number to pick random 1 to 3 if number = 3 then show icon else if number = 2 then show icon else show icon *Example code from Micro:Bit Makecode*
VARIABLES						Understand what **variables** are and how to use them. (orange blocks in Scratch).  Score time set Score to change Score by show variable Score hide variable Score *Example code from Scratch3*	Understand what **variables** are and how to use them. Eg. Use a speed variable to control fairground ride speed up and down  on start set speed to 0 on button A pressed change speed by 1 on button B pressed change speed by -1 forever turn motor 1 forward at speed speed . *Example code from Micro:Bit Makecode*