**A primary progression for programming: Key concepts, skills and approaches to programming**

|  |  |  |
| --- | --- | --- |
| **Computational Thinking Skills For Every Lesson**  Computational thinking is about looking at a problem in a way in which a computer can help us solve it. It is defined as the process of formulating and solving problems by breaking them down into simple steps.  This is a twostep process:  First, we think about the steps needed to solve a problem (algorithm). Then, we use our technical skills to get the computer working on the problem (coding).  Computational thinking is NOT thinking about computers or like computer. <https://www.youtube.com/watch?v=qbnTZCj0ugI> | | |
| reasoning Icon 4776973**LOGICAL REASONING**  **Predicting and analysing**  If you set up two computers in the same way, give them the same instructions (the [program](http://barefootcas.org.uk/barefoot-primary-computing-resources/concepts/programming/)) and the same [input](http://barefootcas.org.uk/programme-of-study/work-various-forms-input/inputs/), you can pretty much guarantee the same [output](http://barefootcas.org.uk/programme-of-study/work-various-forms-output/outputs/). 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. | **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. | decomposition Icon 204135**DECOMPOSITION**  **Breaking down into parts**  The process of breaking down a problem into smaller manageable parts is known as decomposition. Decomposition helps us solve complex problems and manage large projects so they are less daunting and much easier to take on. |
| debugging Icon 4446679**DEBUGGING**  **Finding and fixing errors**  Errors in [algorithms](http://barefootcas.org.uk/sample-resources/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. | evaluating Icon 5484792**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 the quality of the work that they produce – is it fit for purpose? | idea Icon 5838327**TINKERING**  We want to develop in children a willingness to experiment and explore a new app or new software. Children should be encouraged to ‘play’ with a new piece of software, sharing what they discover about it to one another, rather than always coming to the teacher for the answers. Pupils can explore how to use others’ code as a starting point for their own programming projects. Tinkering should help develop independence and perseverance when working with technology. |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Reception** | **Year 1/2 (on a two-year cycle)** | | **Year 3/4 (on a two-year cycle)** | | **Year 5/6 (on a two-year cycle)** | |
| **On screen code** |  | Bee-Bot - Izinhlelo zokusebenza ku-Google Play  Beebot app | Scratch Jr Programming – Damien Kee | Tynker - Crunchbase Company Profile & Funding | Tynker - Crunchbase Company Profile & Funding | Scratch Team - YouTube |  |
| **Physical/applied coding** | Bee-Bots in the Early Childhood Classroom • TechNotes Blog  Beebots | Bee-Bots in the Early Childhood Classroom • TechNotes Blog  Beebots |  |  |  |  | Microbits |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **FS** | **Year 1** | | **Year 2** | | **Year 3** | | | | **Year 4** | | | | **Year 5** | **Year 6** | |
| **SEQUENCING SKILLS** | Sequence forwards and turns e.g. with Beebot  [IMG_2795](https://enablingenvironments.files.wordpress.com/2015/02/img_2795.png)  **Predict** the outcome of a set of instructions and test the results.  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)  4) (clear | **Sequence** commands of forwards, back, left, right using arrow blocks .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.  Understand that a sequence of instructions needs to be clear, precise and unambiguous.  Know how to clear the code. | | **Sequence** commands including forwards, back and turns more efficiently using blocks.  Understand that some steps in a sequence can be reordered but still achieve the same outcome (flexible sequence).  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 numbers of commands.  Understand that a sequence of instructions in computing is called an **Algorithm** and that the instructions for a computer to follow is a **program**.  Use **decomposition** to break the sequence in to manageable steps.  Understand how to approach **debugging** a program or **algorithm**. | | | | Sequence instructions in the correct order to create an animation sequence, draw a shape or solve a problem.  Understand that a sequence of instructions in computing 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. | | | | Describe what **commands**, **functions, debugging** and **sequences** are.  To read code in Swift Code blocks   * Repeat loops * Event handling * Selection   Be able to assess success of given instructions and identify and correct any errors that occur. | To sequence an algorithm using written Swift Code.  To read **and write** Swift code using:   * Repeat loops * Functions * Event handling * Selection * Variables   Be able to evaluate the effectiveness of an algorithm written by their peers in class. | |
| **REPEAT LOOPS**  **(iteration)** |  | |  | | Loop a set of commands by a given amount.  Use a number to specify movement rather than repeated commands (e.g. in Scratch Jnr enter 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 repeats and how many times to achieve a desire end. | | | Describe what **for loops** are.  Use the instruction **repeat until …**    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. | | To read and write **loops.**  Use a **variable and operators** (the green blocks in Scratch) within a loop to govern termination: |
| **EVENT HANDLING SKILLS** | Know that pressing Go will make the robot move.  [IMG_2795](https://enablingenvironments.files.wordpress.com/2015/02/img_2795.png) | | Understand that an **event** is an action that causes something to happen.  Sequence an **event** in words and symbols.  Know that when a key (e.g. space bar) is pressed, the sprite/character will move. | | Express an **event** in words and **symbols**.  Control a character in a game or animation where clicking make something happen. | | Be able to create an animation or game using an existing template or scaffold. | | | | Be able to use a range of **inputs** to start an event or control a character e.g., space bar, mouse click, ipad press.  **Parallelism** – 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). | |  |
| **CONDITIONAL STATEMENTS SKILLS** |  | | | | | | | Understand that we can make actions occur only ​ under certain conditions.  Use **IF statements** in everyday life and in coding | Understand **conditional statements** as a way of ​ handling different situations (using If, Then, Else commands)    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. | | | | Describe what **Conditionals** are.  Use ‘if, then, else’ statements  e.g. in a quiz: if answer correct… | | Describe what **Conditionals** are.  Use selection to govern different events using the ‘if / else’  Eg. Microbit 8 ball | |
| **VARIABLES SKILLS** |  | | | | | | | | | | | Understand **variables** as a way of working with changing values. | | | Describe and understand what **variables** are and how to use them.  Eg. Use a speed variable to control fairground ride speed up and down. | |
| **TINKERING OPPORTUNITIES** | Control a Bee Bot on a floor grid  Control Coji with Emojis  Use Dot and Dash with Go and Path apps | | Control a Bee Bot on a floor grid  Use Scratch JR app | | Control a Bee Bot with Blue-Bot app | | Use Dash robots with Blockly app  Use Hopscotch App | | | | Use Dash robots with Blockly app  Use Sphero with Sphero Edu app | | | Use Lego coding resources  Use Sphero with Sphero Edu app | | Use olly robots  Use Sphero with Swift Playgrounds app |