2.1 – Algorithms - Computational Thinking

What is Computational thinking?
The thought processes involved in formulating a problem and its solution(s), so that a computer, human or machine can effectively carry out

How do you think computationally?
To effectively solve problems you need to:

- Decompose
- Abstract
- Algorithmic thinking
- Create algorithms

KEY TERMS

Algorithm: Steps to provide a solution to a problem, usually represented in flowcharts or pseudocode

Decompose: Breaking down a large problem into smaller sub-problems

Abstraction: Representing 'real world' problems in a computer using variables and symbols and removing unnecessary elements from the problem

Algorithmic Thinking: Identifying the steps involved in solving a problem.

Sequence: Completing steps in the order which they must happen

Selection: Where a choice is made in a program depending on a condition or outcome

Iteration: Act of repeating or lopping specific sections of code

Flowcharts
Displays an algorithm in diagram form using symbols and arrows to show the flow of information

Pseudocode
A structured use of English used to define the steps needed to solve a problem.

WHILE TRUE:
  temp = ""
  IF temp > than 20c THEN
    Open windows AND heaters OFF
  ELSE
    Close windows AND heaters ON
  END IF
  TIME = ""
  IF TIME = 18.00 THEN
    Sprinklers on
  ELSE
    Sprinklers off
  END IF
  BREAK
2.1 – Algorithms – 4. Flowcharts

Sequence that performs a specific task.
You can use this within your flowchart to show more detail in a specific section.

SUB ROUTINE

START/STOP

Always start and end with this.

INPUT/OUTPUT

Use when there is an input or output required e.g. user inputs their name, program displays their name

Decision

Is A>10 ?

When a choice has to be made in the program

YES

No

PROCESS

To do something in the program e.g a calculation.

Flow lines – show the flow of information in the algorithm.
2.1 – Algorithms – 5. Pseudocode

Pseudocode uses English. It mimics how your code may look in the programming language BUT it DOESN’T have to be exact.

Some useful terms you could use.

START
IF
ELIF
ELSE
FOR
WHILE TRUE
ENDIF
END

Here are some symbols to use:

**Comparison operators**

- `==` Equal to
- `!=` Not equal to
- `<` Less than
- `<=` Less than or equal to
- `>` Greater than
- `>=` Greater than or equal to

**Arithmetic Operators**

- `+` Add
- `-` Subtract
- `/` Divide
- `*` Multiply

MOD will return the remainder for the division.

WHILE TRUE:

```
  temp = “”
  IF temp > than 20c THEN
    Open windows AND heaters OFF
  ELSE
    Close windows AND heaters ON
  ENDIF
  IF TIME = “”
    IF TIME = 18.00 THEN
      Sprinklers on
    ELSE
      Sprinklers off
      Continue
    ENDIF
  ENDIF
  BREAK
```

This repeats, it reads the temp of a greenhouse. If greater than 20 degs then open windows and turn heaters off ELSE close windows and heaters on.

It checks the time if the time is 6pm then turn sprinklers on otherwise keep sprinklers off.
2.1 – Sorting Algorithms – Bubble, Insertion & Merge

<table>
<thead>
<tr>
<th>Key vocabulary</th>
<th>Bubble sort</th>
<th>Insertion sort</th>
<th>Merge sort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Works by repeatedly going through the list to be sorted, comparing each pair of adjacent elements. If the elements are in the wrong order, they are swapped. If they are in the correct order, one is left in position.</td>
<td>Sorts data one item at a time. The algorithm takes one data item from the list and places it in the correct location in the list. This process is repeated until there are no more unsorted items in the list. More efficient than bubble sort.</td>
<td>This is a two-stage sort. Firstly, the list is split in half into sublists repeatedly. The algorithm stops splitting the lists when each list has only one element left. The second stage involves repeatedly merging the lists in order until there is only one sublist remaining.</td>
<td></td>
</tr>
</tbody>
</table>

Figure 1 - Bubble sort example

Figure 2 - Insertion sort example

Figure 3 - Merge sort example
2.1 – **Sorting Algorithms - Binary and Linear searches**

### Key vocabulary

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Search</td>
<td>Data may be in any order to complete a linear search. Each item is inspected in turn to see whether it is what is being searched for. If an item is found, then True is returned, else the next element is inspected until all items have been searched. If nothing is found by the end of the algorithm then False is returned.</td>
</tr>
<tr>
<td>Binary Search</td>
<td>If a list is sorted (numerical or alphabetical order) then a more efficient algorithm can be used. It works by repeatedly dividing the list into half and searching in the appropriate half.</td>
</tr>
</tbody>
</table>

**Figure 1 - Linear search example**

**Figure 2 - Binary search example**