Computing fundamentals
Lesson 3: Programming 1

Introduction
In this lesson students develop their understanding of computational thinking further and learn, through practical application, how algorithms are used to create programs, using flowcharts and pseudocode. They also learn about the importance of testing and debugging, giving an opportunity to recap, consolidate and extend learning from KS2.

You will need
Lesson plan, lesson guide, rough paper, micro:bit MakeCode editor

Learning objectives
- To understand the relationship between algorithms and programming
- To understand and use pseudocode and flowchart algorithms
- To tinker, test and debug to create a working program using a graphical programming language

Lesson summary – approx. 60 minutes
1. Recapping algorithms (5 minutes)
2. Pseudocode and flowcharts (7 minutes)
3. Introducing programming (5 minutes)
4. Tinkering with the Make Code editor (10 minutes)
5. Writing programs (10 minutes)
6. Testing and debugging programs (10 minutes)
7. Sharing programs (8 minutes)
8. Review & wrap up (5 minutes)

1. Introduction: Recapping algorithms (5 minutes)
- Give out rough paper, show **slide 2** and ask students to work in pairs to rearrange the instructions to create the algorithm for getting up (there are several possible combinations - a suggestion is on **slide 3**). If appropriate, encourage students to add their own steps too.
- Highlight the repetition used to make the algorithm more efficient, spending more time on this if necessary for your students.
- Ask students to think/pair/share what they know about algorithms from the computational thinking fundamentals lessons 1 and 2 and prior experience to recap, using the discussion to highlight the points on **slide 4**.

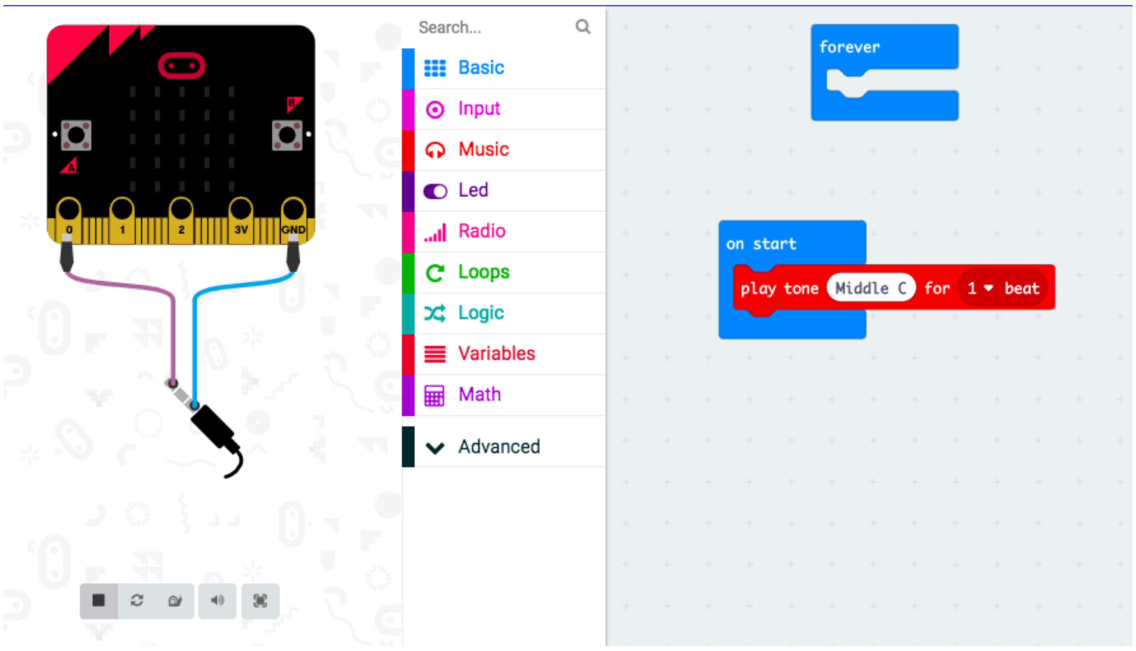
2. Pseudocode and flowcharts (7 minutes)
- Use **slides 5 and 6** to explain (or recap) that while algorithms can just be a simple set of instructions, as in the previous giant paper aeroplane activities, in computing we usually write them using pseudocode and flowcharts.
- Briefly highlight the standard format for pseudocode and standard flowchart symbols and inputs and outputs (these will be returned to in later lessons).
- Invite students to consider why we use pseudocode and/or flowcharts (it makes the algorithm easier for a human to follow when they program it into a computer, is clear and avoids ambiguity, is a standard format, so everyone can follow/understand, can be used with any programming language, improves speed and accuracy etc.)

3. Introducing programming (5 minutes)

- Explain that in this lesson they will be programming using the micro:bit MakeCode editor which is a graphical programming language and share the learning objectives on **slide 7** if you wish.
- Invite students to think/pair/share what they know about programming already (**slide 8**) and which programming languages they have already used or know (**slide 9**), highlighting others which are graphical and have a block-based interface (e.g Scratch, Kodu), which will give them a head start.

4. Tinkering with the MakeCode editor (10 minutes)

- Show students how to access the MakeCode editor and start a new project.
- Individually, or in pairs, give students 5 minutes to 'tinker' with the environment to see what they can find out (try to encourage discovery, rather than giving them instructions, however if they struggle with this, ask them to find out how to create and run a simple program).
- After 5 minutes, invite students to show and share what they have discovered, ensuring you cover as a class how to create, run and stop at least a basic program.



5. Writing programs (10 minutes)

- Show students the algorithm(s) on **slide 10** (printed copies may be useful for them) and ask them to guess what they will do. A flowchart is given just for the first one, so ask students how this will change for the second.
- Explain that their challenge is to create a program from the algorithm using the MakeCode blocks editor.
- Briefly discuss how they can best approach this (e.g. logically, step-by-step), giving additional assistance to any students who may need it.
- Give students 10 minutes to work on the challenge. Those who are confident, can add additional elements to the algorithm and their program.

6. Testing and debugging programs (10 minutes)

- Stop students at an appropriate point and ask them to share any problems they have encountered and how they have approached solving them.
- Highlight that regular testing and finding and fixing errors (debugging) are essential steps in programming, so they need to get in the habit of doing this regularly (**slide 11**).

- Encourage them to help each other to solve problems (i.e. not simply asking you when they are stuck!).
- Give students a further 5 minutes to test and debug their programs.

7. Sharing programs (8 minutes)
- Have a 'round robin' sharing session where students share their programs with their peers.
- Discuss as a class, inviting them to share any different ways they have achieved the same goal, to discuss how they solved any problems they encountered and how they found following the flowchart and/or pseudocode.
- If this would not work owing to space, or dynamics, invite a few students to the front to share and discuss as above.

8. Wrap up (5 minutes)
- Review the learning objectives if you wish on **slide 12** and invite students to answer the questions on **slide 13** either during the lesson or for homework if you wish.

Extension ideas:
- You could ask students to design their own pseudocode and/or flowchart algorithms for everyday activities.
- They could write algorithms for each other, increasing in difficulty, then swap to create, test and debug the programs (this could also be a stretch and challenge activity).

Differentiation

Support:
- Students may benefit from having slides 2,5,6,10 printed out so they can follow more easily.
- They can program only one part of the algorithm to programming activity on slide 10 or set them easier challenges.
- They may benefit from more structured tinkering, being given some guidance as to which coloured blocks to focus on if helpful.

Stretch & challenge:
- Encourage students to create a more complex algorithm and program and their own examples once they have completed the given one.
- Also see extension

Opportunities for assessment:
- Informal observation of students' during activities and discussion.
- Informal assessment of students' programs and answers to review questions.