LESSON 17

Getting Started with Python

80 minutes (2 periods) | Introduction to Python | Outcome 1.22 | Outcome 2.6 | Outcome 2.7 | Outcome 2.20 | Outcome 2.21 | Outcome 3.11 | Outcome 3.14

WHAT THIS LESSON DOES

In this lesson, you'll explore the basics of Python programming using the Micro:bit Python editor. Follow step-by-step instructions to understand Python syntax, write your first program, and experiment with displays and loops on a virtual or physical Micro:bit.

CURRICULUM OUTCOMES

Outcome	What the specification asks for
1.22	read, write, test, and modify computer programs
2.6	construct algorithms using appropriate sequences, selections/conditionals, loops and operators to solve a range of problems, to fulfil a specific requirement
2.7	implement algorithms using a programming language to solve a range of problems
2.20	identify and fix/debug warnings and errors in computer code and modify as required
2.21	critically reflect on and identify limitations in completed code and suggest possible improvements
3.11	use and control digital inputs and outputs within an embedded system
3.14	design automated applications using embedded systems

WHAT STUDENTS SHOULD BE ABLE TO DO

- Understand the basics of Python as a programming language and its applications.
- Familiarise with the Micro:bit Python editor and its interface.
- Grasp the importance of syntax and indentation in Python coding.
- Learn to write and run simple Python programs with comments and sequences.
- Explore basic control structures like loops and display functions in Python.

BEFORE THE LESSON

- Open python.microbit.org yourself and confirm the site and its simulator are not blocked by the school filter. It runs in the browser with no install.
- Type and run the Hello World and Knight Rider snippets in advance so you can spot the indentation errors students will hit.
- If you plan to let students use physical micro:bits, charge or cable them up first. Otherwise the virtual simulator is enough for the whole lesson.

Getting Started with Python

HOW THE LESSON RUNS

- Introduction to Python
- Access the Micro:bit Python Editor
- Understanding Python Syntax and Indentation
- Working with Python Comments
- Write Your First Hello World Program
- Adding Comments to Your Hello World Program
- Understanding Code Sequence
- Errors and Debugging
- Experiment with More Displays
- Send Code to a Micro:bit
- Exercise: Knight Rider
- Extend the Code
- Wrap Up

TEACHING MOVES

- 1. Introduction to Python. Keep this brief. Ask where students have already met Python or heard of it, then move on. The point is to run code, not to read about it.
- 2. Access the Micro:bit Python Editor. Walk them to python.microbit.org together and point out the code pane, the simulator and the Run button. Have them delete the default code but keep the 'from microbit import *' line.
- 3. Understanding Python Syntax and Indentation. Show what happens when you break indentation on purpose. Let the editor throw the error so they see Python enforces it, unlike the braces they may know from JavaScript.
- 4. Working with Python Comments. Stress that everything after the hash is ignored when the code runs. Prove it by commenting out a working line and running it.
- 5. Write Your First Hello World Program. Have everyone type it themselves rather than copy-paste, so they meet the quotes and brackets. Then have them change the message to their own name and run again.
- 6. Adding Comments to Your Hello World Program. Point out the output does not change when comments are added. This confirms the previous step's claim for anyone who doubted it.
- 7. Understanding Code Sequence. Have them predict aloud what order the heart, angry face and message appear before running. Then remove a sleep line and ask why the images now flash past.
- 8. Errors and Debugging. Name the three error types with a live example of each: a missing colon for syntax, an undefined variable for runtime, a wrong sleep value for logical. Let students see the difference between a crash and wrong behaviour.
- 9. Experiment with More Displays. Introduce 'while True:' as forever. Emphasise the indented lines belong inside the loop and ask what would run if a line were un-indented.
- 10. Send Code to a Micro:bit. Only spend time here if you have devices and cables ready. Otherwise say clearly the simulator does everything the course needs and skip to the exercise.
- 11. Exercise: Knight Rider. Draw the 5x5 grid on the board and mark x as column, y as row, both starting at zero. Students consistently swap these, so anchor it before they type set_pixel.
- 12. Extend the Code. Set one change per pair from the list: speed, brightness, row or multiple pixels. Have them run before and after so they can describe exactly what changed.

WHAT TO WATCH FOR

- Students indent inconsistently, mixing tabs and spaces or misaligning lines inside a loop or if block, then cannot see why Python complains. Have them read the error message aloud, then check every line inside a block sits at the same 4-space level. Pick one style and stick to it.
- Students read set_pixel(x, y, value) as row then column, so their pixel appears in the wrong place. Return to the grid on the board. x is across, y is down, both count from 0. Have them point to (0,0) and (4,0) before running.
- Students think code that looks right on the simulator is therefore correct, and stop testing edge cases. Ask them to predict the output before every run, then compare. A logical error runs fine but does the wrong thing, so watching, not just running, is the test.

DIFFERENTIATION

- Emerging: Give them the Hello World line already typed and have them change only the message, so they succeed at running code before touching syntax. Pair them with a student whose editor is working so a filter or login problem does not stall the whole period.
- Developing: Ask them to predict what a snippet does before running it, then explain any difference. This moves them from following steps to reasoning about sequence.
- Proficient: Set the Knight Rider extension to bounce the light back and forth smoothly and scroll a message between passes, using variables to control position. Ask them to explain the difference between a syntax, runtime and logical error using their own code as examples, the kind of clear reasoning useful evidence for an Applied Learning Task.