LESSON 19

Control Structures

80 minutes (2 periods) | Introduction to Python | Outcome 1.7 | Outcome 1.22 | Outcome 2.6 | Outcome 2.7 | Outcome 2.20 | Outcome 3.11 | Outcome 3.13

WHAT THIS LESSON DOES

In this lesson, you'll learn the essentials of control structures for programming with the Micro:bit. Explore conditional statements and loops to make decisions and repeat actions, building interactive code through practical, step-by-step activities.

CURRICULUM OUTCOMES

Outcome	What the specification asks for
1.7	develop algorithms to implement chosen solutions
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
3.11	use and control digital inputs and outputs within an embedded system
3.13	develop a program that utilises digital and analogue inputs

WHAT STUDENTS SHOULD BE ABLE TO DO

- Understand the role of conditional statements in making decisions within programs.
- Master the use of nested conditionals for complex decision-making scenarios.
- Apply loops to efficiently repeat actions and process data.
- Utilise nested loops and combined structures for advanced program control.
- Develop skills in integrating conditionals with loops for dynamic coding solutions.

BEFORE THE LESSON

- Confirm the micro:bit MicroPython editor students use actually flashes to a device or simulator, and that cables are present for physical boards.
- Have the running temperature-checker starter code ready to paste, so each new step edits a working program rather than retyping from scratch.
- Test the while True button-checking exercise yourself first, so you can show a live board responding rather than describing it.

Control Structures

HOW THE LESSON RUNS

- Introduction
- Understanding Conditional Statements
- Simple If Statement
- If-Else Statement
- If-Elif-Else Statement
- Nested If Statements
- Introduction to Loops
- For Loops
- While Loops
- Nested Loops
- Using Loops with Conditionals
- Exercise: Decision Maker
- Exercise: Countdown Timer
- Wrap Up

TEACHING MOVES

- 2. Understanding Conditional Statements. Anchor the idea in the umbrella example before any code. Stress that a condition evaluates to True or False, nothing in between, and that the block only runs on True.
- 3. Simple If Statement. Make indentation the headline. Show what happens when the indented line is unindented: Python errors or runs it unconditionally. Point out that with no else, a false condition simply does nothing.
- 4. If-Else Statement. Frame else as the guarantee that the program always does something. Change the temperature value live and re-run so students see both branches fire.
- 5. If-Elif-Else Statement. Stress that only one branch runs, top to bottom, and order matters. Ask what happens if the elif for 20 comes before the if for 30.
- 6. Nested If Statements. Draw the two-level decision as a flow: outer age check, then inner student check only if age passes. Trace each of the three outcomes aloud before running.
- 7. Introduction to Loops. Sell loops by writing display code ten times, then replacing it with one loop. Contrast for (known count) against while (condition-driven) up front.
- 8. For Loops. Make range(5) explicit: it gives 0 to 4, five values, not 1 to 5. This off-by-one is where most for-loop errors start.
- 9. While Loops. Deliberately write a loop that never updates its variable, run it, and let them see it hang. Then fix it by incrementing inside the loop.
- 10. Nested Loops. Slow this down on the 5x5 grid. Show that the inner y loop completes fully for each single x, lighting a whole column before x moves on.
- 11. Using Loops with Conditionals. Use the even/odd check to show the if runs fresh every iteration. Have them predict the output for 0 to 9 before running.
- 12. Exercise: Decision Maker. Point them to while True and is_pressed(), and insist on the sleep(200). Without it the loop spins so fast the display never settles.
- 13. Exercise: Countdown Timer. Remind them str(count) is needed for scroll, and that the loop condition count greater than 0 is what makes Boom appear instead of 0.

WHAT TO WATCH FOR

- Students use = (assignment) where they mean == (comparison) inside a condition. Say the two aloud differently: 'gets' for =, 'is equal to' for ==. Show the error a single = produces in an if.
- Off-by-one with range(): expecting range(5) to include 5 or start at 1. Print list(range(5)) or scroll each value so students see 0 through 4 for themselves.
- A while loop whose condition never becomes false, so the program hangs. Have them point to the exact line that moves the loop toward stopping. If they cannot, that is the bug.
- Believing multiple if branches can all run, rather than one elif chain choosing exactly one. Contrast three separate if statements with one if-elif-else on the same data and compare the output.

DIFFERENTIATION

- Emerging: Keep them on the working temperature checker and only change the number, so they see branches fire before writing any structure themselves. Pair with a partner and have them read each condition aloud as True or False before running.
- Developing: Ask them to trace the nested loop output on paper for a 2x2 grid before running it on the 5x5. Have them explain in words what the countdown loop condition guarantees before coding it.
- Proficient: Challenge them to combine both exercises: a button-driven menu that launches the countdown, evidence they could reuse in an Applied Learning Task. Ask them to rewrite the countdown with a for loop and reversed range, then argue which loop suits the task better.