LESSON 20

Functions and Modules

80 minutes (2 periods) | Introduction to Python | Outcome 2.9 | Outcome 2.7 | Outcome 2.3 | Outcome 1.22 | Outcome 2.20 | Outcome 1.3 | Outcome 3.13

WHAT THIS LESSON DOES

In this lesson, you'll learn the essentials of coding with functions and modules. Explore how to define and use functions for reusable code, work with parameters and return values, and integrate modules to enhance your Micro:bit projects with ease.

CURRICULUM OUTCOMES

Outcome	What the specification asks for
2.9	assemble existing algorithms or create new ones that use functions (including recursive), procedures, and modules
2.7	implement algorithms using a programming language to solve a range of problems
2.3	implement modular design to develop hardware or software modules that perform a specific function
1.22	read, write, test, and modify computer programs
2.20	identify and fix/debug warnings and errors in computer code and modify as required
1.3	solve problems by deconstructing them into smaller units using a systematic approach in an iterative fashion
3.13	develop a program that utilises digital and analogue inputs

WHAT STUDENTS SHOULD BE ABLE TO DO

- Grasp the concept of functions as reusable mini-programs for specific tasks.
- Master the use of parameters and return values to enhance function flexibility.
- Recognise the importance of code reusability to simplify maintenance and debugging.
- Understand the role of modules in extending program capabilities through pre-written code.
- Develop skills in importing and integrating modules with custom functions for practical applications.

BEFORE THE LESSON

- Confirm the micro:bit Python editor opens and flashes to a device for every seat, or that the simulator works if devices are short.
- Test the music and speech exercises yourself first: music and speech need a device (or a simulator that supports them), not just the editor.
- Have micro:bits cabled up and, for the music exercise, check whether your kit needs a speaker or headphones on the pins.

Functions and Modules

HOW THE LESSON RUNS

- Introduction
- Understanding Functions
- Defining a Simple Function
- Functions with Parameters
- Functions that Return Values
- Reusability Example
- Understanding Modules
- Importing and Using Modules
- Import Music
- Exercise: Random Sound
- Exercise: Speech
- Wrap Up

TEACHING MOVES

- 2. Understanding Functions. Anchor the whole idea in one line: write code once, call it many times. Point to display code they would otherwise repeat, and stress the def keyword plus indentation before any typing.
- 3. Defining a Simple Function. Have them notice that defining a function runs nothing on its own. The device stays blank until the function is called by name. This is the commonest point of confusion, so name it now.
- 4. Functions with Parameters. Draw the distinction out loud: the name in the def line is the parameter, the value you pass when calling is the argument. Show the same function scrolling two different messages.
- 5. Functions that Return Values. Contrast return with display.scroll. One hands a value back to your code, the other just shows something. Point out str(result) is needed because scroll wants text, not a number.
- 6. Reusability Example. Run temperature_status with 30 and 20 and let them see one function handle both cases. This is the payoff of the whole first half, so pause and make it explicit.
- 7. Understanding Modules. Reframe: from microbit import * is itself a module import they have used all along. Modules are reusability at a larger scale. Name random, music and speech as what comes next.
- 8. Importing and Using Modules. Show import random gives access to random.randint. Combine it with one of their own functions so they see imported code and custom code sitting side by side.
- 9. Import Music. Flag the hardware need: music plays through pins, so a device with a speaker or headphones is required, not the editor alone. Play one built-in tune inside a function.
- 10. Exercise: Random Sound. Let them attempt it before you show anything. Steer them to random.randint(1, 3) and an if-elif-else picking BA_DING, NYAN or BIRTHDAY. Insist the play code lives in a function they call twice.
- 11. Exercise: Speech. Have them define a function taking a name parameter and calling speech.say, then call it twice with different names to prove reusability. Suggest a sleep between calls so speech does not overlap.
- 12. Wrap Up. Ask a few students to state in one sentence the difference between a parameter and a return value. Confirm they can explain why functions reduce duplication before finishing.

WHAT TO WATCH FOR

- Students think defining a function with def makes it run. Show a def block with no call and flash it: nothing happens. Then add the call line and run again. Make the distinction between defining and calling explicit.
- Confusing parameter and argument, or thinking a function can only handle the one value it was written with. Call the same function twice with different inputs on the board. Name the def-line word as the parameter and the passed-in value as the argument.
- Trying to display a returned number directly and getting an error, because scroll expects text. Point to str(result). Explain that a returned number is data, and it must be converted to text before display.scroll can show it.

DIFFERENTIATION

- Emerging: Give them the working show_message code and ask only to change the message and call it a second time, so they experience calling before writing. Pair them with a partner whose device flashes cleanly if the editor or hardware is fighting them.
- Developing: Ask them to explain aloud what each line of temperature_status does before moving to the module exercises. Have them predict the output of add_numbers(3, 5) before running it.
- Proficient: Ask them to add a parameter to the random-sound function so the range of tunes can be set by the caller. Challenge them to combine functions, a returned value and an imported module in one small program, the kind of reusable code they could reuse and document in an Applied Learning Task.