LESSON 24

Rock-Paper-Scissors Game

40 minutes | Introduction to Python | Outcome 3.11 | Outcome 3.13 | Outcome 3.14 | Outcome 2.6 | Outcome 2.7 | Outcome 2.16 | Outcome 1.22

WHAT THIS LESSON DOES

In this lesson, you'll build an interactive Rock-Paper-Scissors game with your Micro:bit as the opponent. Follow step-by-step instructions to code button inputs, random choices, and display results. Start creating your game now in the Micro:bit Python editor.

CURRICULUM OUTCOMES

Outcome	What the specification asks for
3.11	use and control digital inputs and outputs within an embedded system
3.13	develop a program that utilises digital and analogue inputs
3.14	design automated applications using embedded systems
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
2.16	use data types that are common to procedural high-level languages
1.22	read, write, test, and modify computer programs

WHAT STUDENTS SHOULD BE ABLE TO DO

- Understand the basics of programming a simple interactive game using a Micro:bit.
- Develop skills in using lists to store and manage data effectively.
- Apply conditional logic with if-else statements to determine game outcomes.
- Implement random selection to simulate computer choices in a game.
- Create interactive user experiences using button inputs and display outputs.

BEFORE THE LESSON

- Open python.microbit.org yourself and confirm it loads through the school filter, and that display.scroll and random.choice run in the simulator.
- If using physical boards, have one micro:bit and a working USB cable per student or pair, and confirm the browser can flash a .hex to the device.
- Have the final working code saved somewhere you can reach so you can help a stuck student without retyping from scratch.

Rock-Paper-Scissors Game

HOW THE LESSON RUNS

Introduction to the Game
Import Modules
Define Choices List
Get User Choice
Generate Computer Choice
Determine the Winner
Add Game Loop
Download your Code
Wrap Up and Extensions

TEACHING MOVES

- 2. Import Modules. Point out that running now does nothing visible but should show no red errors. A spelling slip in 'from microbit import *' or 'import random' is the first thing to catch here, before more code hides it.
- 3. Define Choices List. Insist they type the list rather than paste, then get 'Ready' scrolling before moving on. Confirm everyone sees output, so the input steps that follow have a known-good starting point.
- 4. Get User Choice. Draw the button mapping on the board: A is rock, B is paper, both is scissors. Explain that break exits the inner loop the instant a press is caught, and that 'both' must be tested first or it never fires.
- 5. Generate Computer Choice. Show that random.choice picks one item from the list they already built, so the list has a job. Run it a few times so students see the computer choice actually varies.
- 6. Determine the Winner. Ask students to list all nine combinations first, then map them: same is a draw, three cases win, three lose. Let them find that structure before they type it.
- 7. Add Game Loop. Highlight the two nested while True loops. The inner one waits for one press, the outer one restarts the round. Have a student point to where each loop begins and ends.
- 8. Download your Code. Walk round as devices connect. If the browser cannot see the micro:bit, check the cable is data-capable not charge-only, and that they clicked to pair the device.
- 9. Wrap Up and Extensions. Recap that lists, random, if-else, input and output each did a distinct job. Offer the score-variable extension to anyone finished, and have them save a .hex before they leave.

WHAT TO WATCH FOR

- Ordering the button check with single-button cases before the both-buttons case, so 'both pressed' can never be reached. Trace what happens when both are held: the elif for button A catches it first. Fix by testing 'button_a and button_b' at the top.
- Assuming a game that scrolls a result is therefore correct, without checking the winner logic across all combinations. Have students force specific matchups and confirm the verdict each time. A game that looks lively can still declare the wrong winner.
- Confusing = and == inside the winner comparisons. Remind them one = stores a value, == asks a question. Ask which they want when checking if user_choice equals computer_choice.

DIFFERENTIATION

- Emerging: Give them the button mapping written down and let them get just the input step scrolling 'You: rock' before adding anything else. Pair them with a student whose 'Ready' step already runs, so they start from working code.
- Developing: Ask them to explain in their own words why break is needed and what the sleep(100) does before they add the winner logic.
- Proficient: Set the accelerometer 'shake to play' input as a Higher Level extension, or a running score kept in variables across rounds. Have them write a comment block explaining their winner logic, the kind of design note that supports Coursework Project evidence.