LESSON 28

# Reaction Time Tester Game

40 minutes | Introduction to Python | Outcome 1.22 | Outcome 2.6 | Outcome 2.7 | Outcome 2.8 | Outcome 2.9 | Outcome 3.11 | Outcome 3.13

## WHAT THIS LESSON DOES

In this lesson, you'll create a fun game to test your reaction speed using a Micro:bit. Follow step-by-step instructions to build the game, measure your response time, store high scores, and display results with engaging challenges.

## 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 |
| **2.8** | apply basic search and sorting algorithms and describe the limitations and advantages of each algorithm |
| **2.9** | assemble existing algorithms or create new ones that use functions (including recursive), procedures, and modules |
| **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 concept of reaction time measurement using timers and loops in programming.
- Develop skills in storing and managing data using lists for high scores.
- Apply conditional logic through functions and if statements to compare and rank data.
- Master string formatting to display results clearly on a Micro:bit device.
- Build a functional game that integrates core Python concepts in a practical, engaging way.

### BEFORE THE LESSON

- Open python.microbit.org yourself and run a two-line micro:bit program end to end so you know the editor loads and the simulator works behind the school filter.
- If physical micro:bits are used, charge or cable them up and confirm at least one downloads a .hex over USB.
- Type and run the full finished program once yourself so you can spot where a student's indentation or missing line will break it.

# Reaction Time Tester Game

### HOW THE LESSON RUNS

| |
|---|
| Introduction to the Game |
| Import Modules |
| High Scores List and Game Loop |
| Add a Random Delay |
| Get Reaction Time |
| Store Reaction Time in List |
| Create Function to Check if High Score |
| Display High Scores |
| Download your Code |
| Wrap Up and Extensions |

### TEACHING MOVES

- 2. Import Modules. Point out that running this does nothing visible but should show no error. A red error here means the imports are wrong, so fix it now before any game logic is layered on top.
- 3. High Scores List and Game Loop. Stress that the empty list and the two nested while loops set the whole structure. Have students name what the inner while not button_a.is_pressed() loop is waiting for before they type it.
- 4. Add a Random Delay. Explain random.randint(1000, 5000) is milliseconds, so one to five seconds. This unpredictability is the point: without it students could time the press in advance and the reaction test would be meaningless.
- 5. Get Reaction Time. Draw out that start_time is captured after the image shows, and reaction_time is running_time() minus start_time. Ask them why the subtraction gives elapsed time, not a clock reading.
- 6. Store Reaction Time in List. Confirm every attempt is appended, not just good ones. The list grows each round; the trimming to top three happens later in the function, so do not expect it here.
- 7. Create Function to Check if High Score. Walk through the logic in words first: fewer than three scores means auto-add, otherwise compare against the slowest and replace it. Faster means a lower number, which trips students up every time.
- 8. Display High Scores. Highlight that sorting is ascending so the fastest is rank one. Have students predict the scroll order before running it, then check against the device.
- 9. Download your Code. For those with hardware, watch that the .hex actually transfers and the LED display responds to the real buttons. Simulator-only students can still complete every step.
- 10. Wrap Up and Extensions. Ask each student to name which Python concept did which job. Offer the averaging or accelerometer extension only to those with a fully working game.

**WHAT TO WATCH FOR**

- Students think a higher reaction time number is better because it looks bigger. Reinforce that reaction time is elapsed milliseconds, so a lower number is faster and better. Sort ascending and put rank one on the shortest time.
- Confusing where start_time is set, leading to negative or huge readings. Show that the timer must start immediately after the image appears, not before the delay. Trace one round on the board with sample millisecond values.
- Indentation drift as the nested loops and function are layered on, breaking the program silently. Have students paste nothing and type each block, checking the indent of every while and if. Read the editor's error line together rather than guessing.

**DIFFERENTIATION**

- Emerging: Pair with a working simulator and let them get to the end of step 5, a runnable timer, before attempting the list and function. Give the running_time() subtraction as a labelled diagram so they can match the two lines to it.
- Developing: Ask them to explain the is_high_score function back in plain English before moving to the display step. Have them predict the scrolled output for three known times, then verify.
- Proficient: Set the averaging or accelerometer-input extension, and ask them to justify their design choice as they would in an Applied Learning Task write-up. Challenge them to handle a tie in reaction times without breaking the top-three rule.