LESSON 26

Compass Navigator

40 minutes | Introduction to Python | Outcome 3.11 | Outcome 3.12 | Outcome 3.13 | Outcome 3.14 | Outcome 2.6 | Outcome 2.7 | Outcome 1.22

WHAT THIS LESSON DOES

In this lesson, you'll build a digital compass with your Micro:bit, using its magnetometer to point north and show directional arrows. Learn to calibrate it, map headings, and use coding concepts like functions and control structures.

CURRICULUM OUTCOMES

Outcome	What the specification asks for
3.11	use and control digital inputs and outputs within an embedded system
3.12	measure and store data returned from an analogue input
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
1.22	read, write, test, and modify computer programs

WHAT STUDENTS SHOULD BE ABLE TO DO

- Understand the functionality of a digital compass using a Micro:bit's magnetometer.
- Develop skills in calibrating hardware components like a compass through coding.
- Apply control structures and data structures to map directional data effectively.
- Enhance programming proficiency with functions and string manipulation.
- Create a portable, interactive device displaying directional information.

BEFORE THE LESSON

- Open python.microbit.org and confirm it loads through the school filter, this is the editor the whole lesson uses.
- Run the calibration and heading code yourself in the simulator first, and find the compass slider tool so you can show students how to simulate a change in direction.
- If using physical micro:bits, charge or cable up enough for the class and check a spare, calibration needs the board tilted to fill the LED grid.

Compass Navigator

HOW THE LESSON RUNS

- Introduction to the Compass Project
- Import Modules and Basic Setup
- Calibrate the Compass
- Get and Display Heading
- Define Directions and Arrows
- Map Heading to Direction
- Display Arrow Image
- Download your Code
- Wrap Up and Extensions

TEACHING MOVES

- 3. Calibrate the Compass. Stress the no copy-paste rule out loud and hold to it. Typing each line is where the function structure and indentation sink in. Explain calibration exists because every magnetometer reads slightly differently.
- 4. Get and Display Heading. Point out heading returns 0 to 359 and 0 is north. Show str() converting the integer so display.scroll can take it. Demonstrate the compass slider in the simulator to prove the number changes.
- 5. Define Directions and Arrows. Ask why arrows is a tuple and directions a list. Land the answer: the eight images never change so a tuple protects them, direction handling can stay a list. This is examinable reasoning, not decoration.
- 6. Map Heading to Direction. Draw the 360 degree circle split into eight 45 degree wedges on the board. Show why north wraps: heading under 22 or over 338 both mean north. Trace one boundary value together before they run it.
- 7. Display Arrow Image. Connect the index from the previous function to arrows[index] on the display. Add the button A recalibration check and explain why letting the user recalibrate on demand matters for a real device.
- 8. Download your Code. Walk round as boards connect. The usual snags are the wrong cable, a board not mounting as a drive, or the .hex saved to the wrong place. Have students watch the calibration tilt animation before reading north.
- 9. Wrap Up and Extensions. Recap each construct against its job: function for calibration, tuple and list for data, if-elif for mapping, str for messages. Set the beep-on-cardinal extension for anyone finished.

WHAT TO WATCH FOR

- Students think calibration is a once-off setup step they can skip, or they comment out calibrate_compass() and get garbage headings. Show a reading before and after calibration in the simulator. Make the point that an uncalibrated magnetometer gives meaningless numbers, so the call has to run before the loop.
- The north wraparound trips them up: they write a single range for north and lose the 338 to 359 arc. Return to the circle diagram. North straddles 0, so it needs two conditions joined with or. Have them test headings of 10 and 350 and confirm both show N.
- Confusing the list index with the heading value, or expecting arrows[heading] to work. Separate the two clearly. The heading is 0 to 359, the index is 0 to 7. get_direction_index converts one to the other, and only the index can address the tuple.

DIFFERENTIATION

- Emerging: If the editor or calibration is fighting them, pair them with a working simulator and let them get to a scrolling heading number before touching arrows. Give the wedge boundaries as a printed table so they map the if-elif structure to numbers rather than inventing them.
- Developing: Ask them to explain aloud why the north condition uses or before they run it, then to predict what heading 90 will display.
- Proficient: Set the beep-on-cardinal extension, then ask them to justify the tuple versus list choice in a sentence they could defend in the written paper. Challenge them to refactor get_direction_index to compute the index arithmetically instead of eight if-elif branches.