LESSON 44

Documenting your code

40 minutes | Strand 3: Coding at the next level | Outcome 3.6

WHAT THIS LESSON DOES

Take a working micro:bit Python program with unclear variable names and add comments to make it readable. You will rename variables, add documentation, and explain tricky lines without changing what the program does.

CURRICULUM OUTCOMES

Outcome	What the specification asks for
3.6	document programs to explain how they work

WHAT STUDENTS SHOULD BE ABLE TO DO

- Add comments and clear names so another person can understand your code (LO 3.6)
- Explain why documentation matters (LO 3.6)

BEFORE THE BELL

- Open python.microbit.org yourself and type the press-counter through once, so you know how it behaves and can help quickly when pupils mistype it. The code itself is on the step, so pupils type it from their own screens.
- Optional: print a copy of the messy code for pupils who prefer paper.
- Student devices with a browser open at python.microbit.org

Documenting your code

HOW THE LESSON RUNS

4 min	introduction	Start: what we're building
3 min	content	What documentation means
4 min	content	Open the editor and type the messy program
5 min	content	Rename the unclear variables
4 min	content	Add a comment that says what it does
4 min	content	Explain a tricky line
3 min	content	Read it back
6 min	activity	Your turn
3 min	content	Make sense: what worked, what broke
2 min	content	Reflect
2 min	summary	What you covered

TEACHING MOVES

- Emphasise early that renaming and commenting do not change behaviour-the program works exactly the same before and after. This prevents confusion when pupils see an error and think they have 'broken' it.
- Model the rename-all workflow: show how missing a single instance of the old variable name causes an error, and how that error is actually useful feedback that clear names are working.
- When discussing comments, show the contrast between bad ('# add one to score') and good ('# prevents one long press being counted many times'). A comment earns its place only when the code alone would not make the purpose obvious.
- Pair pupils for the 'Your turn' peer read: each reads the other's documented program cold. That peer test is the real check of learning.

WHAT TO WATCH FOR

- Pupils rename the display line but miss one or both uses inside the button blocks, leaving an old variable name that causes a NameError. Solution: get them to search-and-replace or read through line by line after renaming.
- Pupils add a comment on every line, including obvious ones. Solution: steer them with a simple rule: 'A comment earns its place when the code alone would not make it obvious.'
- A stray old variable name or a comment line without its '#' symbol causes a syntax or name error late in the lesson. Solution: teach them to scan the display error, which points to the broken line; read that line and the one above for typos or missing symbols.

DIFFERENTIATION

- Support: provide a template showing where each comment should go and what it should roughly say; let pupils fill in the blanks rather than writing from scratch.
- Support: pair with a stronger coder who can model the peer-read process and give feedback on their documentation.
- Extension: ask pupils to delete any variable or line the program never uses, then re-test to prove it still works; this reinforces that refactoring does not break behaviour.