PYTHON LESSON

# Displaying Output Exercises

# 💻 Exercises: Displaying Output on the Screen

*Ten exercises to practise every way of building output covered in this lesson — `print()`'s own arguments, concatenation, f-strings, `.format()`, and the format-specifier mini-language. Where an exercise doesn't specify a technique, use whichever one produces the clearest code — part of the point is learning to recognise when an f-string is overkill and when it's exactly what's needed.*

### Exercise 1: Three Ways, One Line

Given `city = "Leeds"` and `population = 812000`, print `"Leeds has a population of 812000"` three separate times — once using `+` concatenation, once using `.format()`, and once using an f-string.

**Goal:** Feel the difference in how much manual bookkeeping (`str()` calls, `+` signs, placeholder counting) each approach requires for the exact same output.

→ Solution

### Exercise 2: A Custom Separator

Given the date parts `day = 27`, `month = 7`, `year = 2026`, use a single `print()` call with the `sep` argument (no string building at all) to display `27/07/2026`.

**Goal:** Practise reaching for `sep` instead of manually gluing separators between values — the simplest tool available for this exact job.

→ Solution

### Exercise 3: A Progress Bar on One Line

Write a loop that counts from 1 to 5, printing `"Step 1..."`, `"Step 2..."`, and so on, all on the **same** terminal line (so the final result looks like `Step 1...Step 2...Step 3...Step 4...Step 5...`), then a real newline once the loop finishes.

**Goal:** Practise controlling `end` deliberately across multiple `print()` calls, including remembering the one `print()` at the very end that restores the normal newline.

→ Solution

### Exercise 4: Fix the Crash

This line raises `TypeError`: `print("You have " + 3 + " new messages")`. Explain in a comment *why* it crashes, then rewrite it two ways — once with concatenation fixed properly, and once as an f-string.

**Goal:** Directly confront the concatenation-with-a-number gotcha from this lesson's closing tip box, rather than just reading about it.

→ Solution

### Exercise 5: A Currency-Formatted Total

Given `prices = [12.5, 3.25, 7.995]`, sum them and print the total as a properly rounded currency value with exactly two decimal places and a `£` sign — e.g. `Total: £23.75`.

**Goal:** Practise the `:.2f` format specifier and notice that it *rounds* the third price rather than truncating it.

→ Solution

### Exercise 6: A Right-Aligned Number Column

Given `scores = [7, 82, 145, 6300]`, print each one right-aligned inside a field 6 characters wide, one per line, so that every digit lines up regardless of how many digits the number has.

**Goal:** Practise the `:>N` alignment specifier and see why alignment only becomes visible once values of different lengths are printed one under another.

→ Solution

### Exercise 7: Zero-Padded IDs

Given a list of raw numeric IDs `[7, 42, 391, 5]`, print each one as a 4-digit, zero-padded ticket number — `"Ticket #0007"`, `"Ticket #0042"`, `"Ticket #0391"`, `"Ticket #0005"`.

**Goal:** Practise `:04d` and recognise when zero-padding (rather than space-padding) is the correct choice — IDs, not free text.

→ Solution

### Exercise 8: Reorder with `.format()`

Given `first = "Grace"` and `last = "Hopper"`, use `.format()`'s indexed placeholders (`{0}`, `{1}`) to print `"Hopper, Grace"` **and** `"Grace Hopper"` from the same two variables, without swapping the arguments passed to `.format()`.

**Goal:** Practise indexed placeholders specifically — the one thing `.format()` can do slightly more directly than a plain f-string, since the same arguments can be reused or reordered without repeating the expression.

→ Solution

### Exercise 9: A Centred Banner

Print the word `"MENU"` centred inside a line of 20 characters total, e.g. `-------MENU-------` (use `:^` with a fill character, not manual dash-counting).

**Goal:** Practise the centre-alignment specifier and a custom fill character, since `:^` alone pads with spaces by default.

→ Solution

### Exercise 10: A Small Formatted Table

Given `students = [("Priya", 88.5), ("Tom", 72.0), ("Aisha", 95.25)]`, print a table with each name left-aligned in a 10-character field and each score right-aligned in an 8-character field with one decimal place, then a final line printing the class average to one decimal place.

**Goal:** Combine loop iteration, running totals, and two different alignment specifiers in the same piece of output — the most "real-world" exercise in this set.

→ Solution