

PYTHON LESSON

Displaying Output - Advanced Exercises

💻 Exercises: Displaying Output - Advanced

The companion set of ten exercises for the Displaying Output - Advanced lesson: `join`, unpacking with `*`, `repr`, dynamic widths, `pprint`, `json.dumps`, and the `file` and `flush` arguments. As in the lesson, ***no functions****: exercises 6 and 7 deliberately use the repeated inline chain so you can judge for yourself when a function would have helped.*

Exercise 1: A List, Three Ways

Given `shopping = ["milk", "eggs", "bread", "tea"]`, use `join` to print it (a) as `milk, eggs, bread, tea`, (b) with `|` between the items, and (c) one item per line using a **single** `print()` call. Add a comment explaining why there is no trailing separator in any of them.

Goal: Practise calling `join` on the separator, and see that changing the glue (including making it a newline) is the only thing that changes.

→ Solution

Exercise 2: A Sum You Can Read

Given `scores = [88, 92, 79, 100]`, print `88 + 92 + 79 + 100 = 359`. Both the left-hand side and the total must be computed from the list, so the line would still be right if the list changed. Do it once with a generator expression and once with `map(str, ...)`.

Goal: Meet the "join needs strings" rule in practice, and compare the two standard ways of converting on the way in.

→ Solution

Exercise 3: Unpack It

Given `temps = [18, 21, 19, 24]`, use `print(*temps ...)` (no `join`, no loop, no `str()`) to print: (a) `18 21 19 24`; (b) `18 / 21 / 19 / 24`; (c) the four numbers on four

separate lines. Then use `print(*range(...), "Liftoff!")` to print the countdown `10 9 8 7 6 5 4 3 2 1 Liftoff!` on one line.

Goal: Use the star to turn a collection into separate arguments, and use `sep` to control the glue, with `print()` doing the string conversion.

→ Solution

Exercise 4: Fix the Crashes

Each of these fails: `print(colours.join(", "))` with `colours = ["red", "green", "blue"]`, and `print(", ".join(ids))` with `ids = [101, 102, 103]`. Say in a comment what error each raises and why, then fix the second one in **two** different ways (one using `join`, one not).

Goal: Confront the two commonest `join` mistakes directly: calling it on the wrong object, and giving it non-strings.

→ Solution

Exercise 5: str vs repr

Given `values = ["7", 7, "seven", 7.0, None, True]`, print a table with three columns for each value: how `str` shows it (`{v!s}`), how `repr` shows it (`{v!r}`), and its type name. Use a header row and a line of dashes, and align the columns with format specs.

Goal: See with your own eyes why `print()` on a collection shows quotes, and how `!r` distinguishes `"7"` from `7`.

→ Solution

Exercise 6: A Neater Record

Given `profile = {"username": "mkhan", "level": 7, "score": 1520.5, "active": True, "badges": ["early", "helper", "streak"], "coords": (55.95, -3.19), "prefs": {"theme": "dark", "alerts": False}}`, print each key and value in this style:

keys padded to 10 characters, lists joined with `, `, tuples joined with `/`, the nested dict indented two spaces with one entry per line, and numbers and booleans *without* quotes. Use an `isinstance` chain. No functions.

Goal: Apply the pattern from the lesson snippet to new data and fix its weak points: non-string items in a tuple, and numbers that should not be quoted.

→ Solution

Exercise 7: One Level Too Deep

Given `order = {"id": 5012, "customer": "Dana", "items": ["pen", "ink"], "delivery": {"city": "Leith", "notes": ["ring bell", "fragile"], "fee": 3.5}}`, print it with lists joined by `, ` and the nested dict indented. This time the **inner** dict contains a list, and that list must print joined too, not in bracket form. In a comment, count how many copies of the list-handling code you wrote.

Goal: Feel why the inline loop struggles with depth: every extra level means another copy of the type checks.

→ Solution

Exercise 8: An Inventory Table

Given `stock = {"Widget": (120, 2.5), "Sprocket (large)": (8, 14.75), "Bolt": (5400, 0.05)}` (name mapped to quantity and unit price), print a table with a header and columns: item (left-aligned, width taken from the *longest name*), quantity (right-aligned with a thousands separator), unit price (`£`, 2 decimals) and line value (`£`, thousands separator, 2 decimals). Finish with a dashed line and a total row.

Goal: Combine a dynamic width with `:,` and `:.2f`, so the layout adapts if a longer item name is added later.

→ Solution

Exercise 9: pprint vs json

Given `config = {"server": {"host": "localhost", "ports": (8000, 8001)}, "debug": True, "tags": ["a", "b"], "owner": None}`, print it three ways: with `pprint` (`width=40`, keeping the original key order), with `json.dumps(..., indent=2)`, and as a single line with `json.dumps(..., sort_keys=True)`. In a comment, list at least three differences you can see between the `pprint` and JSON output.

Goal: Know which tool shows what Python holds and which produces text for other programs.

→ Solution

Exercise 10: Two Channels and a Countdown

Given `tasks = [("backup", True), ("upload", False), ("cleanup", True)]`, loop over the tasks and print `OK: name` to normal output for a success, but `FAILED: name` to **standard error** (`file=sys.stderr`) for a failure. Use `flush=True` on every print so the two kinds of line stay in order. Then print a summary using `len()`, and finish with a countdown `3 2 1 Go!` on one line, with `flush=True` and a short `time.sleep(0.2)` between numbers.

Goal: Use `file=` and `flush=`, the two print arguments most people never meet.

→ Solution