Week 2 — Bringing the Station to Life: Decisions, Loops & Log Files

Course: Scientific Programming with Python
Timebox: Tutorial 75 min · Exercise 5 ≤ 30 min · Exercises 6–10 = home tasks for ~1 week

Exercise Sheet

Your rooftop station is running, but it needs *brains*: rules for decisions, repetitive checks, and neat text logs. This week you'll make the station react to conditions, perform repeated tasks, reuse small utilities, and write/read simple files so your team can analyze data later.

Exercise 1 — "Ticket, Please!" (Access Decision)

Type: General · **Estimated time:** 7–9 min

Only authorized staff may open the roof hatch: a person must have a valid **role** (either `tech` or `pi`) **and** a **badge** that has **not** expired (days to expiry ≥ 0). Print a one-line decision for the guard console.

What you must produce. One cell/script that reads **role** (string) and **days_to_expiry** (integer) and prints `ALLOW (role, X days left)` or `DENY (reason)`.

Inputs/Outputs.
• Inputs: role ∈ {`tech`,`pi`,`student`,`guest`}, days_to_expiry ∈ `[-5, 365]`.
• Output: exactly one line with decision and short reason.
• Edge cases: negative days ⇒ expired; roles are **case-sensitive**.

Reflection. What conditions must be true at the same time?
Why this matters. Real systems combine multiple checks into one decision.

Exercise 2 — Calibration Comfort Category

Type: Scientific · **Estimated time:** 7–9 min

Classify air **temperature (°C)**: `<18: "cold"`, `18–26: "ok"`, `>26–32: "warm"`, `>32: "hot"`. Print the category.

What you must produce. Read a number and print exactly one category word.

Inputs/Outputs.
• Input: temp in `[-40, 60]`.

• Output: one of `cold|ok|warm|hot`.
• Edge cases: 18 and 26 count as ok.

Reflection. How did you avoid overlapping or missing boundaries?
Why this matters. Clear ranges prevent flip-flopping categories.

Exercise 3 — Pick the Plausible Reading

Type: Scientific · **Estimated time:** 10–12 min

Two sensors report `ta` and `tb`. Accept values only inside `[-30, 55]`. If both are valid, choose the one **closer to 22**. If neither is valid, print `no reading`.

Hint: Use the function abs() to computer absolute of a number. So, abs(-3) returns 3, and abs(3) also returns 3.

What you must produce. Read two numbers; print one chosen number or `no reading`.

Inputs/Outputs.
• Inputs: `ta`, `tb` in `[-100, 100]`.
• Output: chosen value (as typed field) or `no reading`.
• Edge cases: equal distance to 22 → prefer `ta`.

Reflection. In what order did you check validity and distance, and why?
Why this matters. Sensor validation precedes all analysis.

Exercise 4 — Reusable Line Maker (Percent Formatting)

Type: General · **Estimated time:** 10–12 min

You need a standard log row: `ID, temp_C, humidity_pct, category`. Build a **reusable unit** that takes four inputs and **returns** a single formatted string using Week-1 `f"…"` formatting.

What you must produce. A callable unit that returns the string, plus a short demo printing two sample lines.

Inputs/Outputs.
• Inputs: `ID` (like `CU-PHY-2`), `temp` (float), `hum` (int), `category` (string).
• Output example: `CU-PHY-2, 21.5, 55, ok` (1 dp for temp; integer for humidity).

Reflection. What makes the output "typed" and easy to parse later?
Why this matters. Reuse ensures identical, machine-friendly rows.

Exercise 5 — Mini CSV Cleaner (≤ 30 min) [ignore this one for now]

Type: Scientific · **Estimated time:** ≤ 30 min

A teammate gives **multiple lines** like `temp;humidity`. Build a small cleaner that reads lines **until** you receive `END` and prints a cleaned CSV row for each good line: `ID, temp_C, humidity_pct, ok_flag`, where ok_flag is `True` iff temp in `[18,26]` and humidity in `[20,80]`.

What you must produce. One Python script: first read `ID` (string). Then repeatedly read lines; ignore **blank** lines; if a line isn't numeric or lacks `;`, print a one-line warning and skip it. Print cleaned rows to screen.

Inputs/Outputs.
• Inputs: `ID`, then lines like `21.4;55`, terminated by `END`.
• Output: one CSV line per valid input (temp with 1 dp; humidity as integer).
• Edge cases: surrounding spaces; malformed entries skipped with a warning.

Reflection. How did you keep reading until the sentinel without getting stuck?
Why this matters. Real-world inputs arrive messy; quick cleaning makes them usable.

Exercise 6 — Rooftop Rounds Log (filter file) (home task)

Type: General · **Estimated time:** 30–45 min (home)

Create `rounds.txt` with exactly **5 lines** you choose (each `minute, note`). Then read the file and print only the lines containing the exact word `hatch` or `alarm`.

What you must produce. A script/cell that **writes** the file (once) and then **reads** and filters it.

Inputs/Outputs.
• Input file: your 5 lines.
• Output: only matching lines, unchanged.
• Edge cases: trailing spaces; newline handling.

Reflection. Why is `with open(...)` safer than manual open/close?
Why this matters. Small text filters are common maintenance tools.

Exercise 7 — Longest "OK" Temperature Streak (home task)

Type: Scientific · **Estimated time:** 40–60 min (home)

Read `temps.txt` (one number per line). Print the length of the **longest consecutive** run with values in `[18, 26]`.

What you must produce. A script that prints a single integer (the streak length).

Inputs/Outputs.
• Input: 15–40 lines, each a number in `[-20, 60]` (blank lines may appear).
• Output: one integer ≥ 0.
• Edge cases: blank lines ignored; all out-of-range → `0`.

Reflection. When do you reset the current streak, and why?
Why this matters. Run/streak metrics appear in quality control.

Exercise 8 — Shift Roster Summary (home task)

Type: General · **Estimated time:** 30–45 min (home)

Read `roster.txt` with lines `name,shift` where shift ∈ {`day`,`evening`,`night`}. Print counts per shift and a total line.

What you must produce. A script that prints four lines: `day: X`, `evening: Y`, `night: Z`, `total: N`. Unknown shifts are ignored with a warning.

Inputs/Outputs.
• Input: 6–30 lines.
• Output: four `label: count` lines.
• Edge cases: extra spaces; unknown shifts → warning line.

Reflection. Which repeated step did you identify and reuse to keep code short?
Why this matters. Counting categories is a daily ops task.

Exercise 9 — Tiny Converter with Defaults (home task)

Type: Scientific · **Estimated time:** 30–45 min (home)

Build a small converter that returns both °C→K and °C→°F, with an optional **rounding** parameter defaulting to 1 decimal place. Demonstrate on three values.

What you must produce. A callable unit + short demo printout.

Inputs/Outputs.
• Inputs: a Celsius number; optional rounding digits (default `1`).
• Output example: `K=294.1, F=69.8` for `21.0` (with default rounding).
• Edge cases: negative temperatures handled normally.

Reflection. Where do defaults simplify repeated calls the most?
Why this matters. Defaults keep call-sites uncluttered and consistent.

Exercise 10 — Append-Only Daily Log (read last 3) (home task)

Type: General · **Estimated time:** 45–60 min (home)

Each run should **append** a row to `daily_log.txt`: `day, ID, note`. Afterwards, **read** the whole file and print **only the last 3 lines**.

What you must produce. A script that opens in append mode, writes one line, closes, reopens to read, then prints the last three lines. (Hint: you can track the last three lines during reading without storing the whole file.)

Inputs/Outputs.
• Runtime inputs: `day` (1–31), `ID` text, `note` (no commas).
• Output: last 3 lines as-is; if file has <3 lines, print all.
• Edge cases: missing file on first run → create it.

Reflection. Why separate append and read phases clearly?
Why this matters. Appending logs without corruption is foundational.

Week-2 boundaries. Use decisions (`if/elif/else`), simple loops (sentinel input, file/string iteration, `break/continue/else`), small utilities (definition/call/return/defaults), and **text file I/O** (`open`, `with`, read/write). Keep Week-1 habits for **typed fields** and `%` formatting. **Avoid** data structures beyond strings (no lists/tuples/dicts/sets required) and avoid depending on `range()`.