



University of
Zurich^{UZH}



Faculty of Science

Scientific Programming with Python

(Not) reinventing the Wheel

16 July 2026

Exercises: Useful Modules & Your Own Package

Part 1 is a collection of small, *independent* bites, one per module from the lecture. Find the right tools for the job in the lectures, there is one or more package specifically designed for each task. In Part 2, you create your own package. Download and unzip the material (`reinvent_material.zip`); it contains the data files for part 1 (`data/`), two modules (`scripts/gaussian.py`, `scripts/significance.py`), and an `analysis.py` script that uses them.

Part 1: Useful Modules

1 – Spot the difference. Two versions of a shopping list: (`"bacon"`, `"egg"`, `"ham"`, `"cheese"`) and (`"bacon"`, `"egg"`, `"spam"`, `"cheese"`). Print a readable, line-by-line diff of the two. Then: a user typed `"pathlyb"` – suggest the closest correct module name from the candidates `"pathlib"`, `"difflib"`, `"datetime"`, `"dataclasses"`, `"itertools"`, `"functools"`.

2 – File hunting. In the material folder: find all `.py` files recursively, print each with its size in bytes, and write the list of paths to `listing.txt` – all without ever calling `open()`.

3 – A date to remember. Marie Curie received her second Nobel prize on `"1911-12-10"`. Parse the string, print which weekday that was and how many days ago it is. Then print the current time in Zurich and convert it to New York time.

4 – The four seasons. Define a type `Season` with exactly four allowed values, such that `Season("winter")` works and members print and compare like ordinary strings. What does `Season("sumer")` do?

5 – Classes without boilerplate. Define a `Measurement` class with fields `value`, `error`, `unit`, `date` and a method `relative_error()` – without writing `__init__` yourself. Create two measurements – what do `print` and `==` give you for free?

6 – Counting words. `data/zen.txt` holds the Zen of Python. Print the five most common words. Then group all distinct words by their first letter into a dict of lists – without ever checking whether a key already exists – and print the words starting with `t`.

7 – Loop helpers. Combine the measured values `[4.9, 5.1]` and the simulated `(5.0, 5.2)` into a single list. Find which orderings of the letters `t`, `c`, `a` are in `{"act", "cat", "tac"}` – without typing the orderings yourself. Deal seven students round-robin into three exercise groups. One module from the lecture solves all three.

8 – Fast Fibonacci. Write the naive recursive Fibonacci; measure `fib(35)` with `time.perf_counter`. Make it fast with a single line from the lecture, measure again – speedup? Then build a function `round2` that rounds to two digits – without `def` or `lambda` – and apply it to a list of floats.

9 – A command-line tool. Write a small CLI `stats.py`: any amount of numbers as arguments, prints their maximum by default and the mean with the flag `--mean`. The lecture showed two modules for this – pick whichever you prefer. Admire the generated `--help`.

10 – Configuration files. `data/config.ini` and `data/config.toml` encode the same settings. Read both and print every value *with its Python type*. Compute `n_events + 1` from both. What does each format give you?

11 – Better than print. Set up logging with a format of your choice and level `INFO`, get a module-level logger and emit one message per level from debug to error. Which ones appear? Then additionally write warnings and up to the file `run.log`.

12 – Pretty text, pretty tables. Rewrap the first three Zen aphorisms into width-40 lines, then prefix every line of the result with " # ". Print a table of n , n^2 , $\sqrt{n}$ for $n = 1 \dots 5$ with aligned columns and borders – the lecture showed a small package for exactly this.

13 – Reading the web. Fetch the course page <https://www.physik.uzh.ch/~python/python/> and print the status code – make sure a failed request would not go unnoticed. Parse the shipped snapshot `data/course.html` and print every link text and target – this part works offline. Bonus: run it on the live page instead.

14 – Calling other programs. Run `git --version` from Python with captured output and print it. Then run a failing command (`ls no_such_file`) twice: once so that Python continues silently and once so that it raises an exception – which argument makes the difference?

15 – Browser remote control [bonus]. Only if you have Chrome or Firefox installed: adapt the lecture demo that remote-controls a real browser to open the course page *headless* and print the page title. This is the tool when the approach of exercise 13 only sees an empty JavaScript shell.

Part 2: Build Your Own Package

In the lecture we packaged `unithelper` – now you do the same with a different library, `ministats`. If you would rather package *your own* functions than ours, go ahead – everything below works the same.

Task 1: From Scripts to a Package [basic]

1. Feel the pain first: `python analysis.py` works *inside* the material folder – and from any other directory? Why?
2. Create a project `ministats` with the src layout from the lecture and move the two modules from `scripts/` in (`__init__.py` should expose the functions).
3. Write the minimal `pyproject.toml` from the lecture (build backend `hatchling`; `name` and `version` are the only required metadata).
4. Install it editable: `pip install -e .`
5. Prove the pain is gone: from your home directory, run
`python -c "import ministats; print(ministats.cdf(0, 0, 1))"`
and fix `analysis.py` so it runs from *anywhere* (no `sys.path!`).

Task 2: The Metadata Puzzle [basic]

Your package is used by others – encode the following requirements in `pyproject.toml` (the lecture slides contain everything you need):

- version 0.1.0, supported Python `>=3.11`, MIT license (SPDX expression + a LICENSE file)
- depends on `numpy` with a lower bound – no upper caps anywhere
- optional extra `plot` that pulls in `matplotlib`
- dependency group `test` with `pytest`
- a console script `zscore`, such that `zscore 5.2 5.0 0.1` prints the deviation in standard deviations and the two-sided *p*-value, for example
`z = 2.00 -> two-sided p = 0.0455`
(write a small `main()`; see part 1)

Then check yourself: `pip show ministats` should display your metadata, and `zscore 5.2 5.0 0.1` print the line above. *Note:* after changing `pyproject.toml`, run `pip install -e .` again – metadata is only read at install time.

Task 3: Build a Wheel and Swap It

1. Build distribution files (`pip install build` first):

```
$ python -m build
$ ls dist/
```

2. Look inside: `unzip -l dist/*.whl` – find your code, your entry point and your metadata (`unzip -p dist/*.whl "*/METADATA"`).
3. Swap wheels with your neighbour (mail, USB stick, chat, ...).
4. Install their wheel – `pip install --force-reinstall path/to/their.whl` – and run their `zscore`. Which pieces of information made that work, and where did they travel? (Afterwards, `pip install -e .` in your project brings your own version back.)