Example Class 2 — Objects, packages, Git and tests

C1 Research Computing — Michaelmas 2026
MPhil in Data Intensive Science and MPhil in Computational Biology

Exercises

Shallow copies of objects holding arrays

The class below, in `orbit.py`, holds an orbit of a predator–prey model: the times `t` and the populations `u` and `v`. Each method returns a modified copy.

```
import copy

import numpy as np


class Orbit:
    def __init__(self, t, u, v):
        self.t = np.asarray(t, dtype=float)
        self.u = np.asarray(u, dtype=float)
        self.v = np.asarray(v, dtype=float)

    def scaled(self, k):
        new = copy.copy(self)
        new.u = k * self.u
        return new

    def backwards(self):
        new = copy.copy(self)
        new.u, new.v = self.u[::-1], self.v[::-1]
        return new

    def phase(self):
        new = copy.copy(self)
        new.t = self.u
        return new
```

`copy.copy(x)` makes a new object whose attributes are bound to those of `x`. A basic slice such as `a[::-1]` is a view: a new array that uses the memory of `a`, whose `base` is the array that owns that memory. `np.shares_memory(a, b)` tells whether `a` and `b` share memory.

(a) Give the two lines printed by

```
O = Orbit([0, 1, 2], [1, 2, 3], [4, 5, 6])
S, B, P = O.scaled(10), O.backwards(), O.phase()
print(S.t is O.t, S.u is O.u, B.u.base is O.u, P.t is O.u)
print(B.backwards().u is O.u, np.shares_memory(B.backwards().u, O.u))
```

(b) The program goes on with `S.u[0] = -1`, `B.u[0] = -2`, `P.t[1] = -3` and `S.t[1] = 9`. Give `O.t`, `O.u` and `O.v`.

(c) Give, as one bullet point, why the assignment through `B` changes `O` and the one through `S` does not.

Version specifiers

`~=V` accepts `V` and any later release that keeps every component of `V` but the last: `~=3.4.2` means `>=3.4.2,==3.4.*`. Clauses separated by commas must all hold. Versions compare component by component, a missing one counting as 0, so 3.10 follows 3.9 and equals 3.10.0.

For each requirement below, list which of the matplotlib versions 3.8.0, 3.8.4, 3.9.2, 3.10.1 and 4.0.0 it accepts.

requirement	accepted versions
`matplotlib~=3.8.2`	
`matplotlib~=3.9`	
`matplotlib!=3.9.*`	
`matplotlib>=3.8,<3.10`	
`matplotlib<=3.9`	

The contents of a wheel

The repository of `halflife` holds the files below; Listing 1 is its `pyproject.toml`.

```
halflife/
  pyproject.toml   README.md
  docs/index.md
  scripts/plot_decay.py
  src/halflife/__init__.py   decay.py
  src/halflife/tables/nuclides.csv   sources.txt
  tests/test_decay.py
```

`decay.py` reads its table of half-lives with

```
text = files("halflife").joinpath("tables/nuclides.csv").read_text()
```

where `files`, from `importlib.resources`, returns the directory of the installed package. The command `halflife C-14` prints the half-life of carbon-14.

```toml
[build-system]
requires = ["setuptools>=75"]
build-backend = "setuptools.build_meta"

[project]
name = "halflife"
version = "0.3.1"
description = "Decay of radioactive nuclides"
readme = "README.md"
requires-python = ">=3.10"

[project.scripts]
halflife = "halflife.decay:main"

[tool.setuptools.package-data]
halflife = ["tables/*.csv"]
```

Listing 1: `pyproject.toml` of `halflife`.

At the repository root, `python -m pip wheel . --no-deps -w dist` writes the first wheel below; the second is a NumPy wheel.

```
halflife-0.3.1-py3-none-any.whl
numpy-2.1.0-cp312-cp312-macosx_14_0_arm64.whl
```

A wheel is the archive that `pip install` unpacks into `site-packages`. It holds the modules that setuptools finds in `src/`, the files declared as package data, and a metadata directory, here `halflife-0.3.1.dist-info`. Its file `METADATA` holds the fields of `[project]` and the readme.

Each package-data pattern is a glob relative to its package. The last three tags of a wheel name give the Python version, the ABI (the binary interface of compiled code) and the platform: an operating system, its oldest supported release, and a processor.

(a) Which of these files does the `halflife` wheel contain?

```
halflife/decay.py              tests/test_decay.py
halflife/tables/nuclides.csv   docs/index.md
halflife/tables/sources.txt    scripts/plot_decay.py
README.md                      halflife-0.3.1.dist-info/METADATA
```

(b) Which lines of Listing 1 put `tables/nuclides.csv` in the wheel?

(c) Give one bullet point on what the tags of the NumPy wheel say. Can it be installed into a CPython 3.13 environment on an arm64 Mac under macOS 26? Into a CPython 3.12 one?

(d) In a fresh clone the lines of (b) are deleted, and the wheel is built and installed in a new environment. Which exception does `halflife C-14` raise, and for which file?

Two letters per file

`git status --short` prints one line per changed file: two letters, then the path. The first letter compares the staging area with the last commit, the second the working tree with the staging area: `M` modified, `A` added, `D` deleted, a space unchanged. Untracked files show `??`, lines are sorted by path, and a clean tree prints nothing.

A repository holds one commit, with `model.py` and `notes.md`, and a clean working tree. Give the output of each `git status --short` marked (a) to (e) in Listing 2.

```
$ echo "rate = 0.3" > fit.py
$ git status --short                        # (a)
$ git add fit.py
$ echo "steps = 200" >> fit.py
$ rm notes.md
$ git status --short                        # (b)
$ git commit -m "Add the fit"
$ git status --short                        # (c)
$ git restore notes.md
$ git add fit.py
$ git status --short                        # (d)
$ git commit -m "Set the number of steps"
$ git status --short                        # (e)
```

Listing 2: A session in the repository.

Branches, merges and fast-forwards

Branch `main` holds the commits A, B and C. Branch `fix` starts at A and gets D; branch `docs` starts at C and gets E. On `main`, `git merge docs` and then `git merge fix` succeed without conflicts.

(a) Draw the final commit graph, with an arrow from each commit to each parent, the three branch names and `HEAD`. Call the merge commits M1, M2, and so on.

(b) Give the parents of each merge commit.

(c) How many new commits does each merge make?

Tests that miss a change

The function of Listing 3 splits an array in two, for example into a training half and a test half. The two tests of Listing 4 pass. A mutation of line 3 replaces `len(x) // 2` with `(len(x) + 1) // 2`.

```
1 def halves(x):
2     """Split x in two; for an odd length the second half is longer."""
3     k = len(x) // 2
4     return x[:k], x[k:]
```

Listing 3: `halves.py`.

```
1  import numpy as np
2
3  from halves import halves
4
5
6  def test_lengths():
7      a, b = halves(np.arange(10))
8      assert (len(a), len(b)) == (5, 5)
9
10
11 def test_rejoin():
12     x = np.arange(1000)
13     assert np.array_equal(np.concatenate(halves(x)), x)
```

Listing 4: `test_halves.py`.

(a) How many tests fail after the mutation? Give, as one bullet point, why.

(b) Give a length for which the two versions differ, and the lengths of the halves before and after the mutation.

(c) Give one `assert` that passes before the mutation and fails after it.

Jobs of a workflow file

Listing 5 is the workflow `.github/workflows/checks.yml` of a GitHub repository. A matrix runs its job once for each combination of its values. Jobs without `needs` start at once, in parallel.

A job with `needs` waits until every job it names has succeeded, each job of a matrix included, and is skipped if one fails. With `fail-fast: false`, a failed job of the matrix does not cancel the others.

```yaml
name: checks

on:
  push:
    branches: [main]

jobs:
  test:
    strategy:
      fail-fast: false
      matrix:
        os: [ubuntu-latest, macos-latest]
        python: ["3.11", "3.12", "3.13"]
        numpy: ["numpy>=2.1,<2.3", "numpy>=2.3"]
    runs-on: ${{ matrix.os }}
    steps:
      - uses: actions/checkout@v5
      - uses: actions/setup-python@v6
        with:
          python-version: ${{ matrix.python }}
      - run: python -m pip install ".[test]" "${{ matrix.numpy }}"
      - run: python -m pytest

  lint:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v5
      - run: pipx run ruff check .

  docs:
    needs: [test, lint]
    runs-on: ubuntu-latest
    permissions:
      contents: write
    steps:
      - uses: actions/checkout@v5
      - run: pipx run mkdocs gh-deploy --force
```

Listing 5: `.github/workflows/checks.yml`.

(a) How many jobs does one push to `main` run, if all succeed?

(b) The job with `macos-latest`, Python 3.11 and `numpy>=2.3` fails at its `pytest` step. Which jobs run, and which are skipped?

(c) Give, as one bullet point, what line 31 protects against.