Lecture 12 — From scripts to an installable package

C1 Research Computing — MPhil in Data Intensive Science, Michaelmas 2026

Boris Bolliet

University of Cambridge

Code and material

```
$ git clone https://github.com/borisbolliet/attractor.git ~/GitHub/attractor  # once
$ cd ~/GitHub/attractor
$ git fetch --tags
$ git switch --detach lecture-12-start
$ source setup_env.sh                    # now also installs the project
$ python -m doctest sessions/lecture12/*.txt
$ cd /tmp && attractor-simulate          # a command of the package, from any folder
```

- `git stash` first if `git status` lists local edits.
- `setup_env.sh` installs the lock. Then it installs the project with `pip install -e . --no-deps`.
- `lecture-12` is the state at the end of the lecture: `git diff lecture-12-start lecture-12`.
- `doctest` prints nothing when every output matches.

Outline

- The import system: the search path, packages, relative imports, the `src` layout
- Project metadata: `pyproject.toml`, version specifiers, extras, editable installs after a lock
- Console commands, the public interface and semantic versions
- Wheels: their contents, package data, compatibility tags
- Live demonstration: `attractor` as an installable package, and a wheel that leaves out a data file

An installed package is found through `sys.path` from any folder. `pyproject.toml` declares what to install. The wheel holds only what the declarations name.

The import search path

```
>>> import pathlib, subprocess, sys, tempfile
>>> folder = pathlib.Path(tempfile.mkdtemp())
>>> _ = (folder / "random.py").write_text('print("random.py of this folder")')
>>> run = [sys.executable, "-c", "import random"]
>>> subprocess.run(run, cwd=folder, capture_output=True, text=True).stdout
'random.py of this folder\n'
>>> subprocess.run(run, capture_output=True, text=True).stdout
''
```

1. `import name` searches the folders of `sys.path` in order.
2. It imports the first match. No match raises `ModuleNotFoundError`.

- First entry: the script's folder, or the current folder for `-c`, `-m` and the prompt. Then `PYTHONPATH`, the standard library, `site-packages`.

The entries of `sys.path`

entry	purpose
`.../lib/python313.zip`	optional zip of the standard library, often absent
`.../lib/python3.13`	the standard library: `os`, `json`, ...
`.../lib/python3.13/lib-dynload`	compiled standard modules (`.so`)
`''`	the current folder
`.../.venv/.../site-packages`	packages installed in the environment
`.../attractor/src`	the project, added by the editable install

- In the project root, `import simulate` imports the local `simulate.py`, found through `''`.
- A local file found before `site-packages` replaces the installed module, and Python raises no error.

Packages and relative imports

```
>>> import importlib.util, json, math
>>> hasattr(json, "__path__"), hasattr(math, "__path__")
(True, False)
>>> json.__file__.endswith("json/__init__.py")
True
>>> import json.decoder
>>> json.decoder.__name__, json.decoder.__package__
('json.decoder', 'json')
>>> importlib.util.resolve_name(".decoder", "json")
'json.decoder'
```

- A **package** is a folder with an `__init__.py`. It holds **submodules**, such as `pkg.mod`, and may hold subpackages.
- `__init__.py` runs at the first import of the package or of a submodule.
- `from .mod import f` is a **relative import**. It fails in a module run as a file.

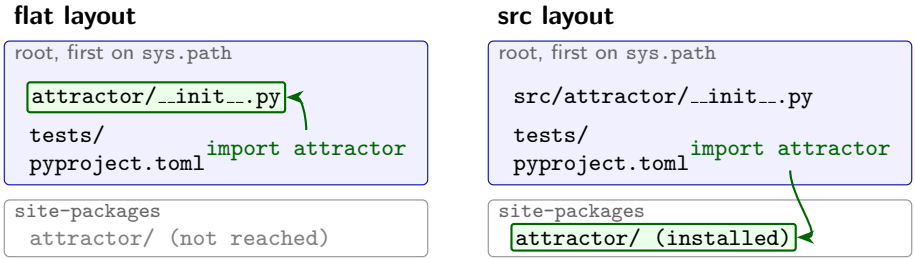
Relative imports at three levels

```
pkg/
├── __init__.py
├── utils.py
└── analysis/
    ├── __init__.py
    ├── fitting.py
    └── models/
        ├── __init__.py
        ├── other.py
        └── linear.py
```

```python
# in pkg/analysis/models/linear.py
from . import other      # pkg.analysis.models
from .. import fitting   # pkg.analysis
from ... import utils    # pkg
```

- One dot names the package of the importing module. Two dots name the parent package, three dots the parent of the parent.

The src layout




```
[tool.setuptools.packages.find]
where = ["src"]
```

- Flat layout: the root's package folder is found, installed or not.
- src layout: the root holds no package. So `import` reaches the installed package, the files a user receives.

Distributions and their metadata

```
>>> import importlib.metadata as md
>>> md.packages_distributions()["dotenv"]
['python-dotenv']
>>> meta = md.metadata("python-dotenv")
>>> meta["Version"], meta["Requires-Python"]
('1.2.4', '>=3.10')
>>> md.requires("python-dotenv")
['click>=5.0; extra == "cli"']
```

- A **distribution** is what pip installs. It holds import packages and their **metadata**. `python-dotenv` installs `dotenv`, and `scikit-learn` installs `sklearn`.
- `pyproject.toml` declares the distribution. `[build-system]` names the **build backend**, the tool that builds it.
- `[project]` holds the metadata. `[tool.name]` holds the settings of one tool.

The project file of `attractor`

```
[project]
name = "attractor"
version = "1.0.1"
description = "Predator-prey models, fixed-step integrators and cycle periods"
readme = "README.md"
license = "MIT"
requires-python = ">=3.12"
dependencies = ["numpy>=1.26"]

[project.optional-dependencies]
fit = ["scipy>=1.11"]
plot = ["matplotlib>=3.8"]
```

- `[build-system]` asks for `setuptools>=77`. pip installs it into a temporary environment for the build: **build isolation**.
- The README becomes the long description. pip refuses Python older than 3.12.

Version specifiers

```
>>> releases = ["1.9.3", "1.10.1", "1.11.4", "1.15.3", "1.18.1"]
>>> specs = ["~=1.11.2", "~=1.11", "==1.15", "==1.15.*",
...          "!=1.11.*", ">=1.10,<1.15", "<=1.15"]
>>> for spec in specs:
...     print(f"{spec:<13}", list(SpecifierSet(spec).filter(releases)))
~=1.11.2      ['1.11.4']
~=1.11        ['1.11.4', '1.15.3', '1.18.1']
==1.15        []
==1.15.*      ['1.15.3']
!=1.11.*      ['1.9.3', '1.10.1', '1.15.3', '1.18.1']
>=1.10,<1.15  ['1.10.1', '1.11.4']
<=1.15        ['1.9.3', '1.10.1', '1.11.4']
```

- `SpecifierSet` comes from `packaging`, the library with which pip reads requirements. The releases are those of SciPy.

Reading a version specifier

specifier	meaning
`~=1.11.2`	`>=1.11.2,==1.11.*`: later fixes of the 1.11 series
`~=1.11`	`>=1.11,==1.*`: every 1.x from 1.11 on
`==1.15`	exactly 1.15.0, so not 1.15.3
`==1.15.*, !=1.11.*`	the 1.15 series, and anything outside the 1.11 series
`>=1.10,<1.15`	both clauses: from 1.10.0 up to 1.15.0, excluded
`<=1.15`	at most 1.15.0, so not 1.15.3

- Versions compare component by component, as numbers. So 1.10 follows 1.9.
- A missing component counts as 0. So 1.15 equals 1.15.0.

Clauses separated by commas must all hold. `~=V` keeps every component of V but the last.

Optional dependencies

```
$ dotenv --help
It seems python-dotenv is not installed with cli option.
Run pip install "python-dotenv[cli]" to fix this.
$ zsh -c "pip install -e .[plot]"
zsh:1: no matches found: .[plot]
```

- An **extra** is a named group of requirements for one feature. In the metadata: `click>=5.0; extra == "cli"`.
- `attractor` has two extras: `fit` for SciPy and `plot` for Matplotlib. `cli.py` imports each inside the function that needs it.
- zsh reads `[plot]` as a pattern of file names. Quote it: `pip install -e ".[fit,plot]"`.

An editable install

```
$ source setup_env.sh
setup_env.sh: installing the project (pip install -e . --no-deps)
[...]
attractor       1.0.0
attractor file  ~/GitHub/attractor/src/attractor/__init__.py
[...]
$ cat .venv/lib/python3.13/site-packages/__editable__.attractor-1.0.0.pth
~/GitHub/attractor/src
```

1. `pip install -e .` writes the metadata into `site-packages`.
2. It writes a `.pth` file that names `src/`.
3. At start-up, `site` adds that folder to `sys.path`.

- Edited code acts at the next run. Version, requirements and commands change only at a reinstall.

A lock, then the project without dependencies

```
$ python -m pip install --dry-run "numpy<2.5"      # the lock holds NumPy 2.5.4
Collecting numpy<2.5
  Downloading numpy-2.4.6-cp313-cp313-macosx_14_0_arm64.whl.metadata (6.6 kB)
Would install numpy-2.4.6
```

- pip keeps a locked version that satisfies a requirement, and replaces one that does not.
- First `pip install -r requirements-lock.txt`. Then `pip install -e . --no-deps`, the project alone.
- `pip check` reports a requirement left unmet.

After a lock, `--no-deps` keeps the tested versions of every dependency.

Console commands

```
>>> (ep,) = md.entry_points(group="console_scripts", name="dotenv")
>>> ep.value, ep.module, ep.attr
('dotenv.__main__:cli', 'dotenv.__main__', 'cli')
```

- An **entry point** names an object of a package, `module:attribute`. A line `name = "pkg.module:function"` of `[project.scripts]` makes `name` a **console command**.

1. At install time, pip writes the **launcher** `name` into `bin/`.
2. The launcher imports the module and calls the function, without arguments.
3. It passes the return value to `sys.exit`: `None` gives status 0.

✎ The commands of `attractor`

```
[project.scripts]
attractor-simulate = "attractor.cli:simulate"
attractor-periods = "attractor.cli:periods"
attractor-fit = "attractor.cli:fit"
```

```
$ cat .venv/bin/attractor-simulate
#![...]/attractor/.venv/bin/python
import sys
from attractor.cli import simulate
if __name__ == '__main__':
    [...]
    sys.exit(simulate())
$ cd / && attractor-simulate
801 points, step 0.05, algae period 8.16 days
```

- `simulate(argv=None)` parses `sys.argv`, or a list passed from Python.
- The launcher is plain Python, so the command runs from any folder.

The public interface

```
>>> import json
>>> json.__all__
['dump', 'dumps', 'load', 'loads', 'JSONDecoder', 'JSONDecodeError', 'JSONEncoder']
>>> "codecs" in vars(json), "codecs" in json.__all__
(True, False)
>>> names = {}
>>> exec("from json import *", names)
>>> sorted(name for name in names if name != "__builtins__")
['JSONDecodeError', 'JSONDecoder', 'JSONEncoder', 'dump', 'dumps', 'load', 'loads']
```

- The **public interface** is the set of names that users may rely on, listed in `__all__`. Names with a leading underscore are private.
- `__init__.py` imports them from the submodules, so users write `package.name`.
- The namespace of `json` holds `codecs`. The public interface does not: `json.codecs` exists, but it is not public.

Semantic versions

- `MAJOR.MINOR.PATCH` states what a release does to the public interface. A break increments MAJOR, an addition MINOR, a fix PATCH. The components to the right reset to 0.
- NumPy 2.0.0 removed `np.NaN`: a major release. NumPy 1.26.4 is the fourth release of fixes to 1.26.
- `~=1.26` relies on the convention: every 1.x from 1.26 on, and no 2.x.
- A published version is never reused for other files. So a fixed wheel needs a new number.

The version is written once, in `pyproject.toml`. `__version__` reads it from the installed metadata.

The contents of a wheel

```
>>> print(dist.read_text("WHEEL").strip())      # dist = distribution("python-dotenv")
Wheel-Version: 1.0
Generator: setuptools (84.0.0)
Root-Is-Purelib: true
Tag: py3-none-any
```

```
$ cat .venv/lib/python3.13/site-packages/numpy-2.5.4.dist-info/WHEEL
[...]
Root-Is-Purelib: false
Tag: cp313-cp313-macosx_14_0_arm64
```

- A **wheel** is a zip archive that pip unpacks. It holds the packages, the declared package data, and `name-version.dist-info`: `METADATA`, `WHEEL`, `RECORD`, `entry_points.txt`.
- Tests, documentation, scripts outside the package and the README file stay out.

The wheel of `attractor` 1.0.0

```
$ python -m pip wheel . --no-deps -w dist -q
$ unzip -Z1 dist/attractor-1.0.0-py3-none-any.whl
attractor/__init__.py
attractor/cli.py
[...]
attractor/trajectory.py
attractor-1.0.0.dist-info/licenses/LICENSE
attractor-1.0.0.dist-info/METADATA
attractor-1.0.0.dist-info/WHEEL
attractor-1.0.0.dist-info/entry_points.txt
attractor-1.0.0.dist-info/top_level.txt
attractor-1.0.0.dist-info/RECORD
```

- The eight modules and the metadata, and no `data/lynx_hare.csv`: nothing at `lecture-12-start` declares it.

Package data

```
>>> from importlib.resources import files
>>> rc = files("matplotlib").joinpath("mpl-data/matplotlibrc")
>>> rc.is_file(), rc.read_text().splitlines()[0]
(True, '#### MATPLOTLIBRC FORMAT')
>>> files("matplotlib").joinpath("mpl-data/no_such_file").is_file()
False
```

- **Package data** are files of a package that are not modules.
- A module goes into the wheel because it lies in the package.
- A data file goes in only when `[tool.setuptools.package-data]` declares it.
- `importlib.resources.files("pkg")` is the folder of the installed package.

A table missing from the wheel

```
$ python3.13 -m venv ../fresh
$ ../fresh/bin/python -m pip install -q "dist/attractor-1.0.0-py3-none-any.whl[fit]"
$ ../fresh/bin/attractor-fit
Traceback (most recent call last):
  File "~/GitHub/attractor/../fresh/bin/attractor-fit", line 7, in <module>
    sys.exit(fit())
[...]
  File "~/GitHub/fresh/lib/python3.13/site-packages/attractor/datasets.py", line 18, in lynx_hare
    text = files("attractor").joinpath("data/lynx_hare.csv").read_text()
[...]
FileNotFoundError: [Errno 2] No such file or directory: '[...]/site-packages/attractor/data/
    lynx_hare.csv'
```

- The editable install runs `attractor-fit` without error. It reads the table from `src/`, where the file lies.

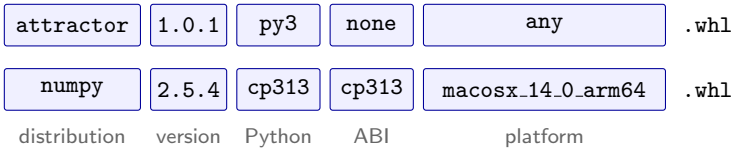
Only a wheel installed into a new environment shows a missing file.

The fix, in version 1.0.1

```
$ git diff lecture-12-start lecture-12
[...]
-version = "1.0.0"
+version = "1.0.1"
[...]
+[tool.setuptools.package-data]
+attractor = ["data/*.csv"]
$ python -m pip wheel . --no-deps -w dist -q
$ unzip -Z1 dist/attractor-1.0.1-py3-none-any.whl | grep csv
attractor/data/lynx_hare.csv
```

- In the new environment `attractor-fit` now prints `LotkaVolterra(a=0.481, b=0.0248, c=0.0275, d=0.926)`.
- A fix that changes no name is a patch release. The editable install reports 1.0.0 until `source setup_env.sh` installs it again.

Compatibility tags



- Python: `py3` or `cp313`. ABI, the binary interface of compiled code: `none`, `cp313`, or `abi3`, the stable subset. Platform: `any`, or macOS 14.0 or later on arm64.

1. The interpreter accepts an ordered list of tags. CPython 3.13 on the lecture laptop accepts 683 (`pip debug --verbose`).
2. pip installs a wheel only if one of its tags is in the list.
3. It prefers the wheel whose tag comes first. `py3-none-any`, a wheel of Python code alone, comes last and fits everywhere.

Which interpreter installs which wheel

```
$ python -m pip install --dry-run ../wheels/numpy-2.5.4-cp312-cp312-macosx_14_0_arm64.whl
ERROR: numpy-2.5.4-cp312-cp312-macosx_14_0_arm64.whl is not a supported wheel on this platform.
$ python -m pip install --dry-run --no-deps dist/attractor-1.0.1-py3-none-any.whl
Processing ./dist/attractor-1.0.1-py3-none-any.whl
Would install attractor-1.0.1
$ ../old/bin/python -m pip install --dry-run --no-deps dist/attractor-1.0.1-py3-none-any.whl
Processing ./dist/attractor-1.0.1-py3-none-any.whl
[...]
ERROR: Package 'attractor' requires a different Python: 3.10.18 not in '>=3.12'
```

- `python` is the project's CPython 3.13. The NumPy wheel compiled for CPython 3.12 needs the ABI of 3.12.
- `py3-none-any` installs into any Python 3 that `requires-python` admits. `../old` holds Python 3.10.

Summary

- `import` takes the first match along `sys.path`. A package in `src/` is found only once it is installed.
- `pyproject.toml` names the build backend and the metadata. A version must satisfy every clause of a specifier.
- An editable install puts `src/` on the path. Code edits act at once. Metadata edits act at a reinstall.
- `[project.scripts]` become launchers. `__all__` and the semantic version state the interface.
- A wheel holds the packages, the declared package data and `dist-info`. Its tags decide which interpreters install it.