Lecture 4 — Reproducible Python environments

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

Boris Bolliet

University of Cambridge

Outline

- Project files that record an environment
- Package constraints, installed versions and compatibility
- Runtime settings in environment variables and `.env`
- Rebuilding and checking an environment
- pip, conda, Colab, imports and Jupyter kernels

Project files record environment choices

Environment component	Project record
Python interpreter	`setup_env.sh` selects `python3.12` by default
Package constraints	`requirements.txt`
Installed package versions	`requirements-lock.txt`
Runtime settings	Tracked template `.env.example`; local file `.env`

borisbolliet/attractor | venv

Create a project environment

```
$ python3.12 -m venv .venv                  # create, once per project
$ source .venv/bin/activate                 # each new shell
(.venv) $ python -m pip install -r requirements.txt
(.venv) $ deactivate
```

- `python3.12 -m venv .venv` creates the environment directory.
- Activation puts its `python` and `pip` commands first on `PATH`.
- `python -m pip` installs into that environment; `-r` reads package requirements from a file.

venv | pip install

✎ Package constraints and installed versions

```
(.venv) $ python -m pip freeze > requirements-lock.txt
```

```
# selected requirements
numpy>=1.26
matplotlib>=3.8
python-dotenv
ipykernel
```

```
# pip freeze snapshot
# Python 3.12.12, macOS arm64
matplotlib==3.9.4
numpy==1.26.4
python-dotenv==1.2.4
...                    (52 packages)
```

- `requirements.txt` permits a range of versions.
- `python -m pip freeze` writes the installed distributions to `requirements-lock.txt`.

requirements file format | pip freeze

✎ NumPy API compatibility

```
$ python3.13 -m venv .venv313 && .venv313/bin/pip install -r requirements.txt
(.venv313) $ python analyse.py
AttributeError: ‘np.NaN‘ was removed in the NumPy 2.0 release.
Use ‘np.nan‘ instead.

# In analyse.py, replace np.NaN with np.nan
(.venv313) $ python analyse.py
...
C9          141      17           8.059
...
```

- `requirements-lock.txt` names the versions the script was run with: Python 3.12, `numpy==1.26.4`. Installing from it reproduces that run; the constraint `numpy>=1.26` alone admits 2.x.

NumPy 2.0 migration guide

Per-run environment variables

`analyse.py` reads `RESULTS_PATH`; when unset, it writes to `results/`.

```
$ RESULTS_PATH=/tmp/attractor-out python analyse.py
...
written: /tmp/attractor-out/periods.csv, /tmp/attractor-out/periods.png
```

- `RESULTS_PATH=value command` sets the variable for that command and its child processes.
- Python reads it with `os.environ.get("RESULTS_PATH", "results")`; the second argument is the default.

os.environ

Local settings in `.env`

```
# Example contents of .env.example
DATA_PATH=data
RESULTS_PATH=results
```

```
$ cp .env.example .env
$ set -a; source .env; set +a
```

- Commit `.env.example` as a template. Copy it to the ignored `.env` for local settings.
- `set -a` exports plain assignments while sourcing; `set +a` turns automatic export off afterward.
- Alternatively, prefix each assignment with `export`; then `source .env` needs no `set -a`.

set | source

Load .env in Python

```python
# the mechanism of check_env.py, in the attractor/ project directory
from dotenv import load_dotenv
import os

load_dotenv()
print(os.getenv("DATA_PATH", "data"))
# output: data
```

```
(.venv) $ DATA_PATH=/tmp/data python check_env.py
...
DATA_PATH       /tmp/data
...
```

- `load_dotenv()` searches from the script's directory upwards; here it reads `attractor/.env`.
- The shell value `/tmp/data` overrides `DATA_PATH=data` in `.env` for this run.

python-dotenv | os.environ

Setup script rebuilds the environment

```
$ source setup_env.sh
```

- Creates `.venv` with Python 3.12 when the environment is absent.
- Installs packages from `requirements-lock.txt` when the snapshot has changed and registers a Jupyter kernel.
- Loads `.env` and runs `check_env.py` to report the configuration.

setup_env.sh in the repository

Sourcing changes the current shell

```
$ ./setup_env.sh; echo "exit: $?"
setup_env.sh: use 'source setup_env.sh'; ...
exit: 1
$ source setup_env.sh
```

- Running a script starts a child process. Changes to its `PATH` disappear when it exits.
- `source` runs the commands in the current shell, so activation remains in effect.

Bash builtins: source

✎ Same setup on macOS and Ubuntu

```
$ source setup_env.sh
version         3.12.12
numpy           1.26.4
...
$ ssh orion
orion$ source setup_env.sh
version         3.12.3
numpy           1.26.4
...
```

- One setup script runs on both systems.
- Both environments install NumPy 1.26.4 and produce the same analysis table.

attractor

Course examples use pip

- The course uses Python's `venv` module to create environments and `pip` to install packages.
- conda is an alternative environment and package manager; it can also install non-Python dependencies such as MPI or HDF5.

Google Colab runtime

```
# Colab notebook cell, after connecting to a hosted runtime
!git clone https://github.com/borisbolliet/attractor.git
%cd attractor
import sys
print(sys.version)
%pip install -r requirements.txt
```

- `Runtime > Change runtime type` selects an available past image with its bundled Python; it cannot install an arbitrary Python.
- `%pip` installs into the hosted interpreter; Colab does not create a project `venv`.
- Runtime files are temporary; mount Google Drive for files that must persist.

Google Colab | runtime versions | local runtimes | files and Google Drive

PYTHONPATH and imports

```
$ export PYTHONPATH=$HOME/old_code
(.venv) $ python -c "import numpy; print(numpy.__file__)"
Example output:
/home/user/old_code/numpy/__init__.py
$ unset PYTHONPATH
```

- Python searches `PYTHONPATH` directories before the environment's installed packages.
- Here, importing `numpy` selects the older copy in `old_code`.

PYTHONPATH

Jupyter kernel and Python interpreter

```
# In a notebook cell:
import sys
print(sys.executable)
# Example output:
/path/to/attractor/.venv/bin/python
```

- A Jupyter kernel is the process that executes notebook cells.
- Its kernelspec names the Python executable; this one points to the `attractor` virtual environment.

Jupyter kernels | sys.executable

Summary

- `requirements.txt` states allowed package versions; `requirements-lock.txt` records one installed set.
- `setup_env.sh` rebuilds the virtual environment from those files.
- Environment variables and `.env` configure a particular run.
- A notebook kernel runs code with its Python interpreter; Colab supplies a hosted one.