

Python packaging, Part 1

Guillaume Witz

Data Science Lab, University of Bern

DSL

Program

Morning 9.00 - 12.00

- Creating a package
- Testing code

Lunch 12.00 - 13.00

Afternoon: 13.00 - 17.00 (or whenever I run out of material)

- Release on PyPi
- Automated testing and release

The Python scientific ecosystem

... A **programming language** and a way to execute it ...



... **Specific tools** for our scientific task (packages) ...



... A solution to **interact** with code and data ...



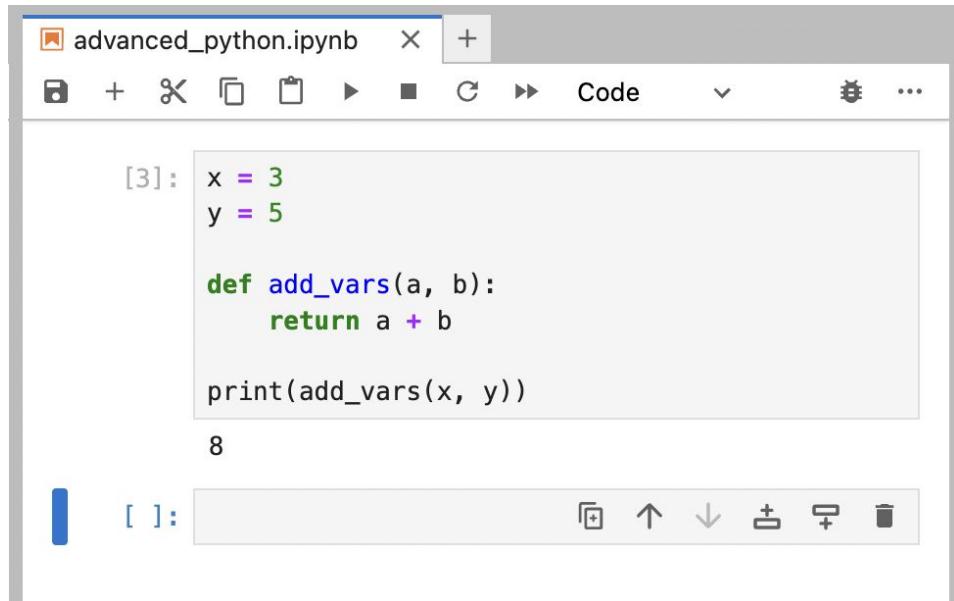
... **Infrastructure** to install and run the code



Executing scripts / modules

Executing Python

Interactively e.g. Ipython, notebook,
command line



The screenshot shows a Jupyter Notebook interface with the title bar "advanced_python.ipynb". The code cell [3] contains the following Python code:

```
x = 3
y = 5

def add_vars(a, b):
    return a + b

print(add_vars(x, y))
```

The output of the cell is "8". Below the cell, there is an empty input cell starting with "[]:".

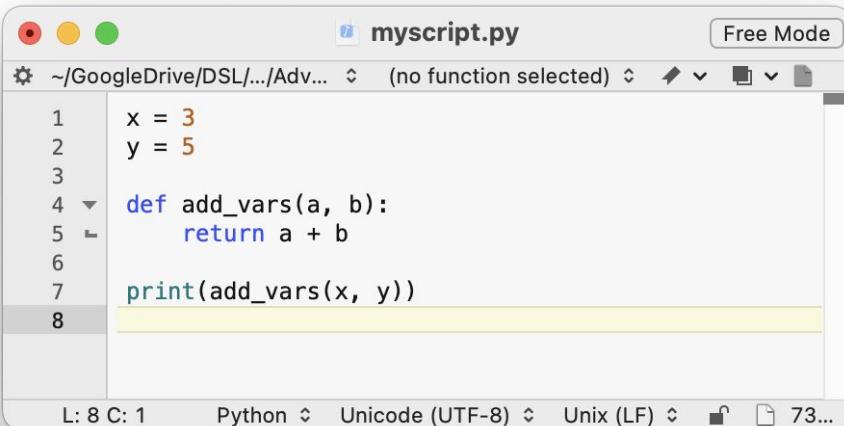
Executing Python

Interactively e.g. Ipython, notebook,
command line

Executing Python

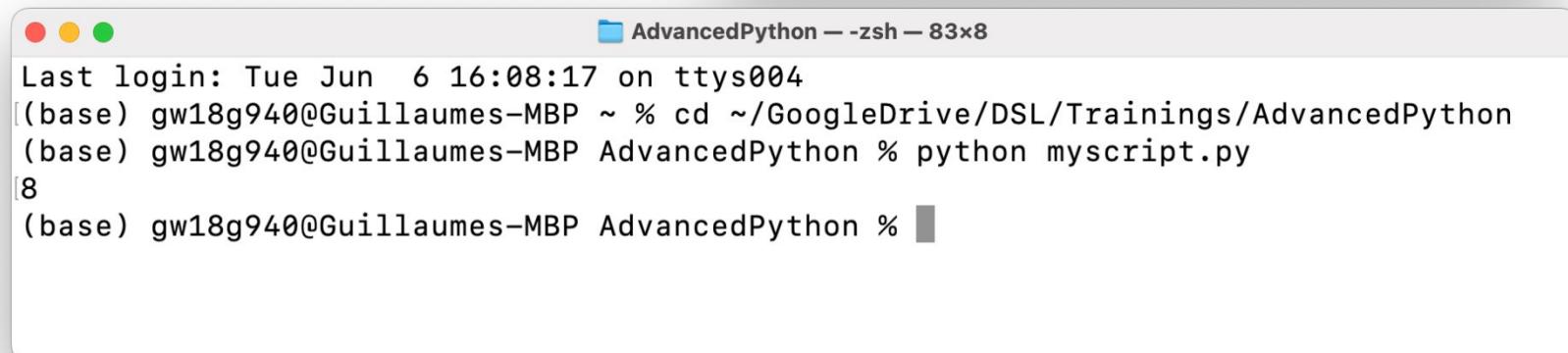
Interactively e.g. Ipython, notebook, command line

Write a script, execute in terminal



```
myscript.py
Free Mode
~ /GoogleDrive/DSL/.../Adv... (no function selected)
1 x = 3
2 y = 5
3
4 def add_vars(a, b):
5     return a + b
6
7 print(add_vars(x, y))
8

L: 8 C: 1 Python Unicode (UTF-8) Unix (LF) 73...
```



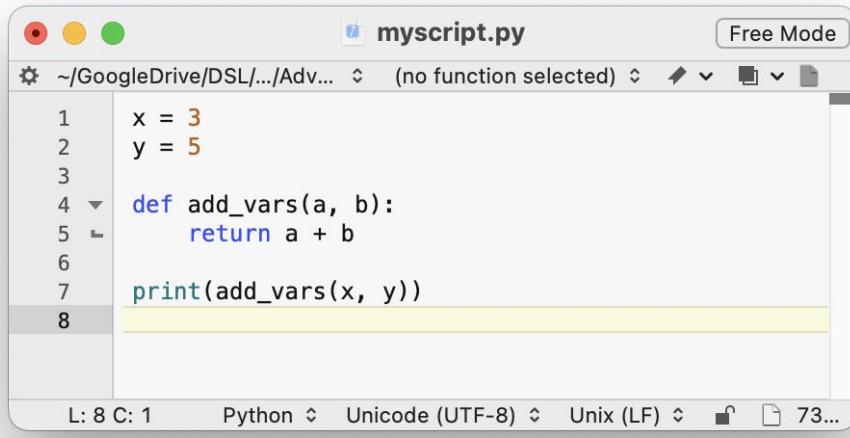
```
Last login: Tue Jun  6 16:08:17 on ttys004
(base) gw18g940@Guillaumes-MBP ~ % cd ~/GoogleDrive/DSL/Trainings/AdvancedPython
(base) gw18g940@Guillaumes-MBP AdvancedPython % python myscript.py
8
(base) gw18g940@Guillaumes-MBP AdvancedPython %
```

Executing Python

Interactively e.g. Ipython, notebook, command line

Write a script, execute in terminal

Import the script as a module:
problem: everything gets executed
at import time



```
myscript.py
Free Mode
~ /GoogleDrive/DSL/.../Adv...
(no function selected)
1 x = 3
2 y = 5
3
4 def add_vars(a, b):
5     return a + b
6
7 print(add_vars(x, y))
8

L: 8 C: 1 Python Unicode (UTF-8) Unix (LF) 73...
```

```
[4]: import myscript
```

```
8
```

```
[5]: myscript.add_vars(3,4)
```

```
[5]: 7
```

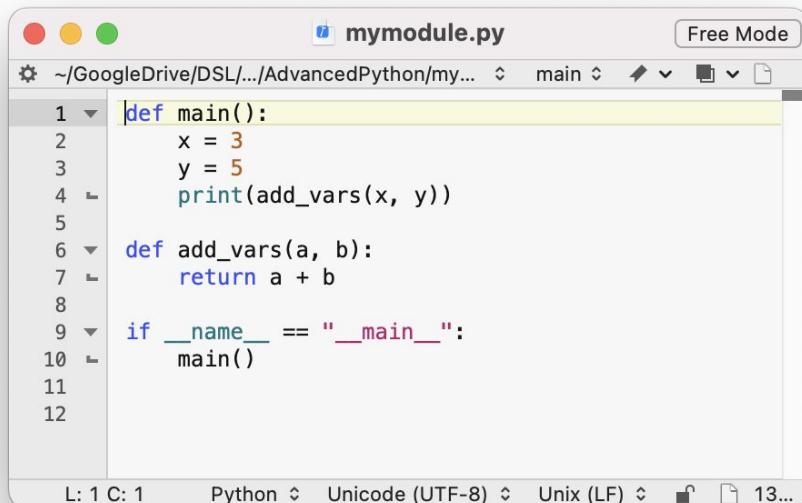
Executing Python

Interactively e.g. Ipython, notebook, command line

Write a script, execute in terminal

Import the script as a module:
problem: everything gets executed
at import time

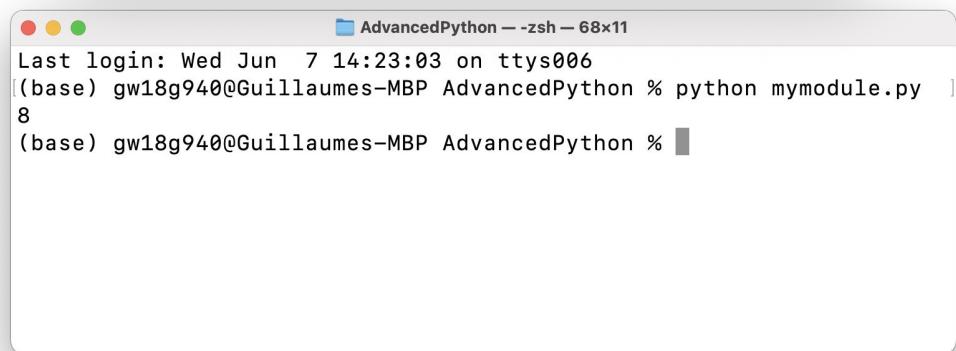
Adding a main() file and checking
the context of execution (`__name__`)
executes main() only when used as
script



```
def main():
    x = 3
    y = 5
    print(add_vars(x, y))

def add_vars(a, b):
    return a + b

if __name__ == "__main__":
    main()
```



```
Last login: Wed Jun  7 14:23:03 on ttys006
(base) gw18g940@Guillaumes-MBP AdvancedPython % python mymodule.py
8
(base) gw18g940@Guillaumes-MBP AdvancedPython %
```

Executing Python

Interactively e.g. Ipython, notebook,
command line

Write a script, execute in terminal

Import the script as a module:
problem: everything gets executed
at import time

Adding a main() file and checking
the context of execution (`__name__`)
executes main() only when used as
script

```
[4]: import myscript
```

8

```
[5]: myscript.add_vars(3,4)
```

```
[5]: 7
```

```
[6]: import mymodule
```

```
[7]: mymodule.add_vars(3,4)
```

```
[7]: 7
```

First problem

Your module is not in the same folder as your current working directory: unfindable

Possible to add to path

Better solution...

The screenshot shows a JupyterLab interface. On the left is a file browser window titled 'notebook_sub... - JupyterLab' showing a directory structure under 'AdvancedPython / subfolder /'. A file named 'notebook_subfold...' is selected. On the right is a code editor window titled 'notebook_subfolder.ipynb' with a Python 3 (ipykernel) tab. The code cell [1] contains 'import mymodule'. This results in a 'ModuleNotFoundError' with a traceback. The user then adds 'sys.path.append('..')' to the code, which fixes the import. Subsequent cells show the module being imported and its 'add_vars' method being called.

```
[1]: import mymodule
-----
ModuleNotFoundError
  ...
Cell In[1], line 1
----> 1 import mymodule

ModuleNotFoundError: No module named 'mymodule'

[2]: import sys
[3]: sys.path
[3]: ['/Users/gw18g940/GoogleDrive/DSL/Trainings/AdvancedPython/subfolder',
      '/Users/gw18g940/mambaforge/envs/advpython/lib/python310.zip',
      '/Users/gw18g940/mambaforge/envs/advpython/lib/python3.10',
      '/Users/gw18g940/mambaforge/envs/advpython/lib/python3.10/lib-dynload',
      '',
      '/Users/gw18g940/mambaforge/envs/advpython/lib/python3.10/site-packages']
[4]: sys.path.append('..')
[5]: import mymodule
[6]: mymodule.add_vars(3,4)
[6]: 7
[ ]:
```

Beyond scripts: packaging

More than one module?

Re-use in other folder? Re-use by other people?

Sure that someone using the module has all required dependencies?

Want to test your code on multiple platforms, with multiple Python versions?

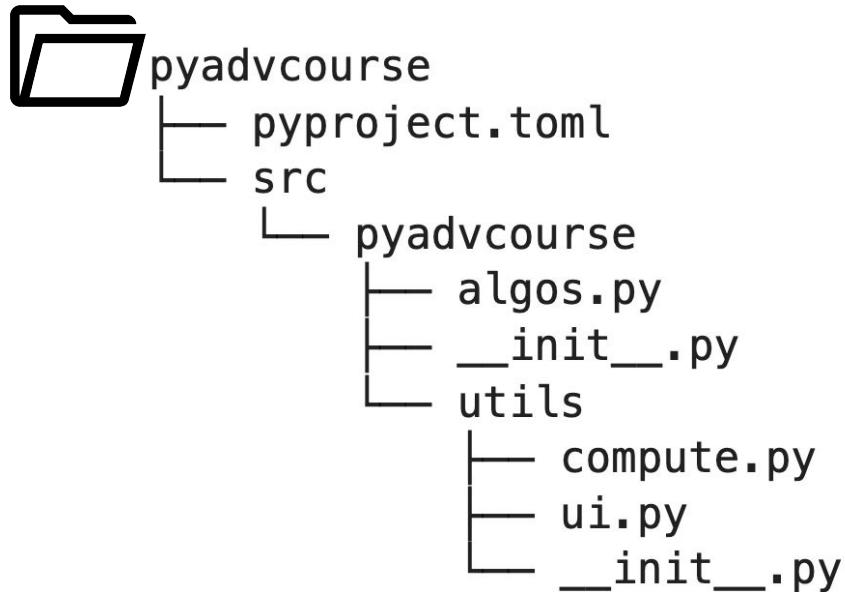
Packaging and distributing
(it's a mess 😱)

The screenshot shows a web browser window with the following details:

- Title Bar:** "How to improve Python packaging, or why fourteen tools are at least twelve too many" - chriswarrick.com/blog/2023/01/15/how-to-improve-pyth...
A red "DSL" logo is in the top right corner.
- Header:** Kw Chris Warrick Contact Projects Guides Archives Search Polski
- Section Header:**

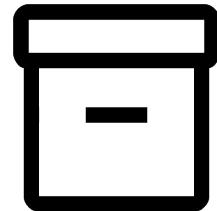
How to improve Python packaging, or why fourteen tools are at least twelve too many
- Date:** 15 January 2023 at 14:45 (updated 18 April 2023 at 00:45)
- Tags:** Python .NET CSharp JavaScript Node.js npm packaging PDM pip PyPA Python virtual environments
- Text:** A large block of text discussing the challenges of Python packaging, mentioning the variety of tools like setuptools, conda, Pipenv, Poetry, Hatch, and PDM, and the confusion it causes beginners regarding virtual environments.
- Text:** A summary of the post's content, explaining the journey through various packaging stacks and tools, and the look at modern workflows and the PyPA's vision.
- Contents Sidebar:** A teal sidebar containing a "Contents" section with a list:
 - The plethora of tools
 - The classic stack
 - ...and a few extensions
- Link:** <https://chriswarrick.com/blog/2023/01/15/how-to-improve-python-packaging/>

Packaging and distributing



Folder with modules, infos about package and dependencies

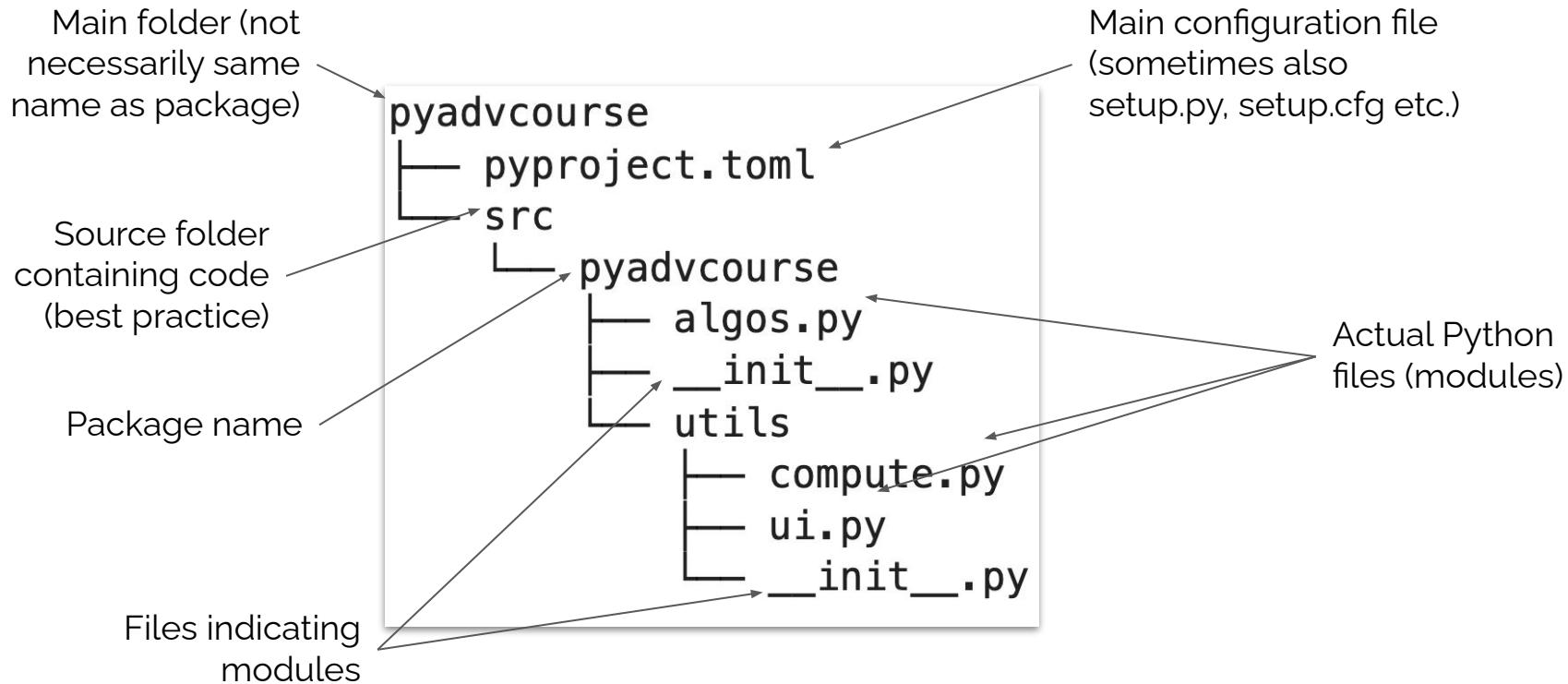
Building process



Built package to be:

- used by others
- uploaded to PyPi

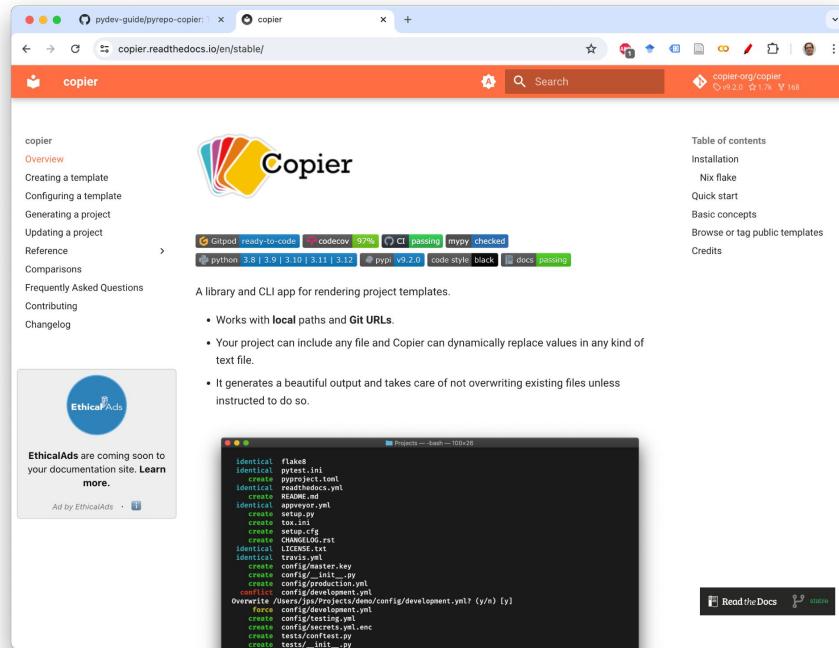
Package structure



Creating a package

Instead of doing this by hand: cookiecutter

Tools like cookiecutter and copier allow to create entire projects populated with relevant files with good defaults.

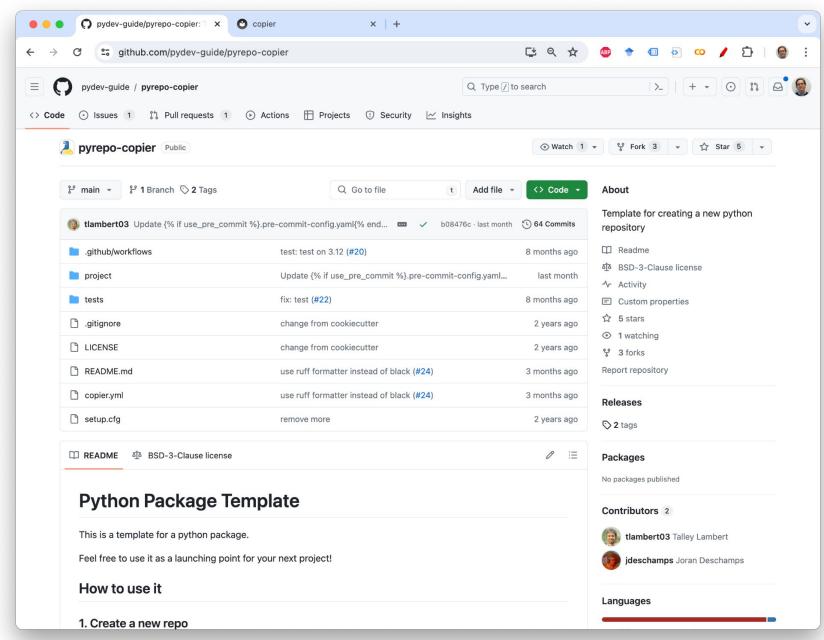


The screenshot shows the Copier documentation page. At the top, there's a navigation bar with links for 'copier', 'Overview', 'Creating a template', 'Configuring a template', 'Generating a project', 'Updating a project', 'Reference', 'Comparisons', 'FAQ', 'Contributing', and 'Changelog'. Below the navigation is a large logo for 'Copier' with a colorful, abstract design. To the right of the logo is a 'Table of contents' sidebar with links for 'Installation', 'Nix flake', 'Quick start', 'Basic concepts', 'Browse or tag public templates', and 'Credits'. The main content area contains a brief introduction: 'A library and CLI app for rendering project templates.' It lists several features: 'Works with local paths and Git URLs.', 'Your project can include any file and Copier can dynamically replace values in any kind of text file.', and 'It generates a beautiful output and takes care of not overwriting existing files unless instructed to do so.' At the bottom, there's a code snippet showing the generated directory structure:

```
Projects --- bash -- 100/26
├── identical
│   ├── flake8
│   ├── pytest.ini
│   ├── README
│   ├── setup.py
│   ├── tox.ini
│   └── CHANGELOG.rst
├── identical
│   ├── .gitignore
│   ├── LICENSE
│   ├── README.md
│   ├── copier.yml
│   └── setup.cfg
└── config
    ├── development
    │   ├── config
    │   │   └── development.yml
    │   └── tests
    │       └── confest.py
    └── production
        ├── config
        │   └── production.yml
        └── tests
            └── _int_.py
```

At the bottom right is a 'Read the Docs' button.

<https://copier.readthedocs.io/en/stable/>



The screenshot shows the GitHub repository for 'pyrepo-copier'. The repository has 1 branch, 2 tags, and 64 commits. The repository description is 'Template for creating a new python repository'. It includes sections for 'Readme', 'BSD-3-Clause license', 'Activity', 'Custom properties', '5 stars', '1 watching', '3 forks', 'Report repository', 'Releases' (2 tags), 'Packages' (No packages published), 'Contributors' (2 contributors: tlambert03 and jdeschamps), and 'Languages' (Python). The repository has 1 pull request, 1 issue, and 1 action. The 'Code' tab is selected, showing the repository structure and commit history. The commit history includes changes from cookiecutter to pyrepo-copier for various files like README, LICENSE, and setup.py.

<https://github.com/pydev-guide/pyrepo-copier>

Create a package

Create an environment and install the cookiecutter package:

```
conda create -n mypackaging python=3.10 jupyterlab copier -c conda-forge  
conda activate mypackaging
```

Then move to where you want to create the folder containing your package

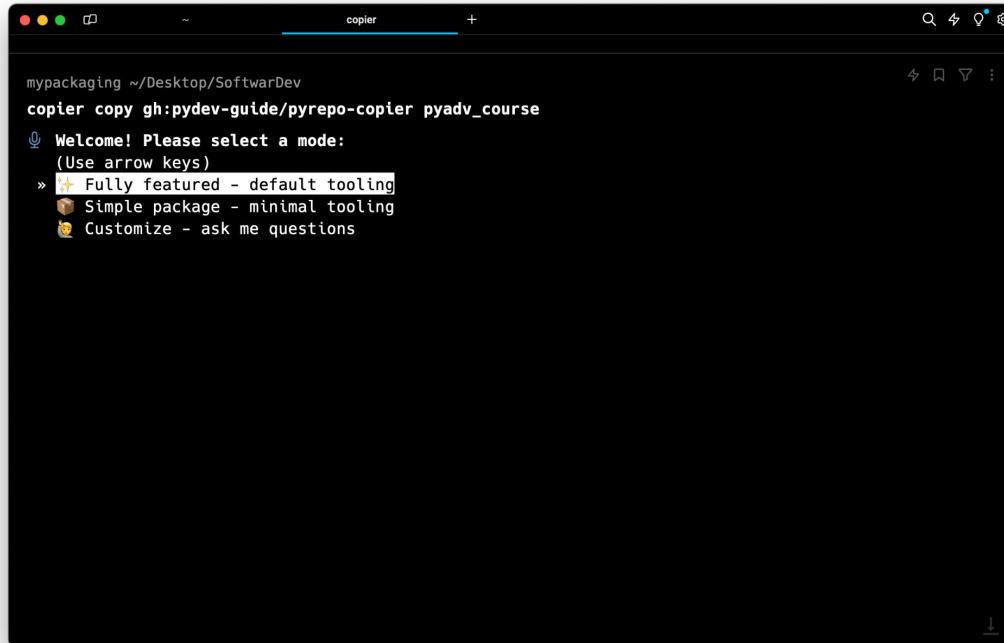
```
cd /to/my/favorite/place
```

Use copier with pyrepo-copier:

```
copier copy gh:pydev-guide/pyrepo-copier your-package-name
```

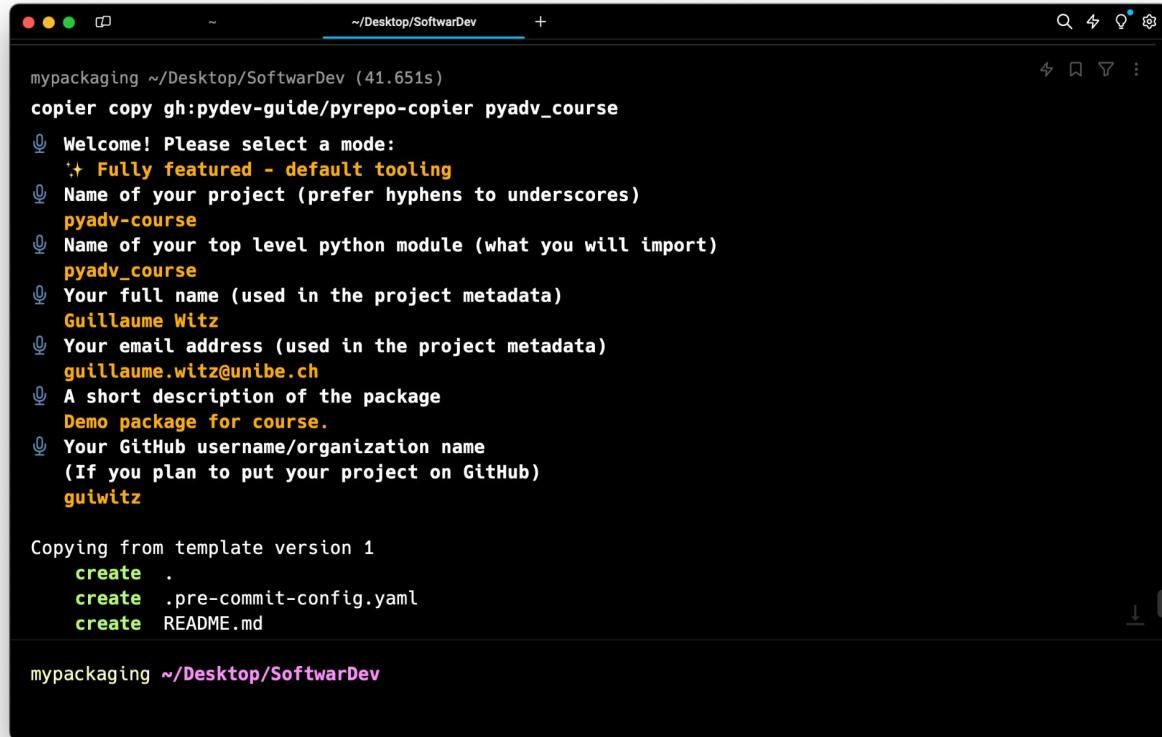
copier package

copier walks you through package creation by asking you a set of questions.
Let's use "Fully featured" even if we won't be using everything.



```
mypackaging ~/Desktop/SoftwarDev
copier copy gh:pydev-guide/pyrepo-copier pyadv_course
➊ Welcome! Please select a mode:
(Use arrow keys)
» 🏙 Fully featured - default tooling
📦 Simple package - minimal tooling
👤 Customize - ask me questions
```

copier package



A screenshot of a terminal window titled "mypackaging ~/Desktop/SoftwarDev". The window shows the execution of the command "copier copy gh:pydev-guide/pyrepo-copier pyadv_course". The terminal then displays a series of questions and answers for creating a new Python package:

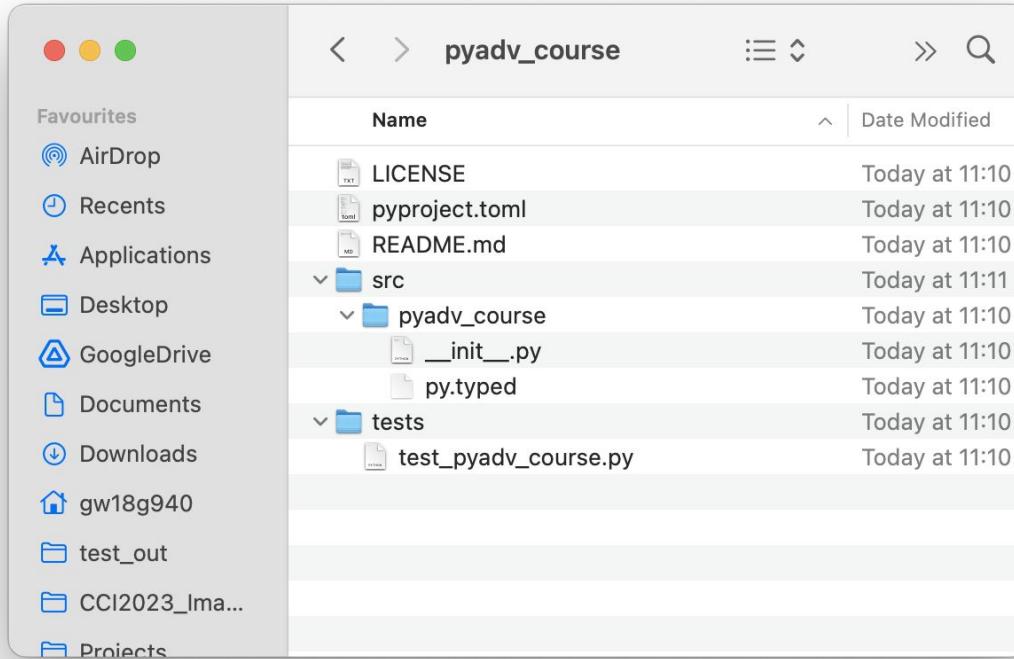
- Welcome! Please select a mode:
:+ Fully featured - default tooling
- Name of your project (prefer hyphens to underscores)
pyadv-course
- Name of your top level python module (what you will import)
pyadv_course
- Your full name (used in the project metadata)
Guillaume Witz
- Your email address (used in the project metadata)
guillaume.witz@unibe.ch
- A short description of the package
Demo package for course.
- Your GitHub username/organization name
(If you plan to put your project on GitHub)
guiwitz

Copying from template version 1

```
create .
create .pre-commit-config.yaml
create README.md
```

mypackaging ~/Desktop/SoftwarDev

copier result



Tour of contents

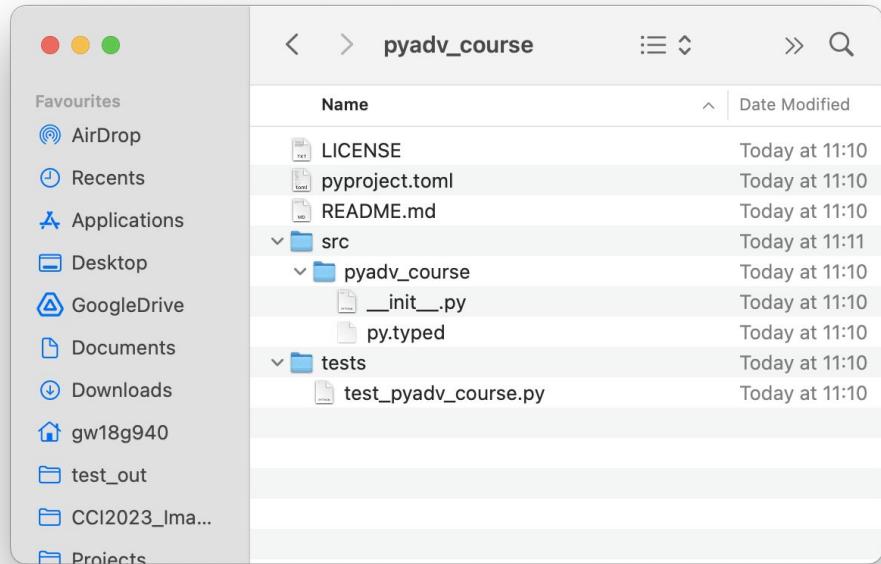
LICENSE: Markdown file to edit

README.md Markdown file to edit
as you want

src: folder with your code. Empty for
now

tests: single python file with pass
function

pyproject.toml: package
configuration



More infos on building

[https://www.pyopensci.org/python-package-guide/package-structure-code
/python-package-build-tools.html](https://www.pyopensci.org/python-package-guide/package-structure-code/python-package-build-tools.html)

pyproject.toml

Sections of the file

Some sections are necessary
(build-system, project)

Some depend on the tooling used
(tool.hatch.version)

```
# https://peps.python.org/pep-0517/
[build-system]
requires = ["hatchling", "hatch-vcs"]
build-backend = "hatchling.build"

# https://hatch.pypa.io/latest/config/metadata/
[tool.hatch.version]
source = "vcs"

# read more about configuring hatch at:
# https://hatch.pypa.io/latest/config/build/
[tool.hatch.build.targets.wheel]
only-include = ["src"]
sources = ["src"]

# https://peps.python.org/pep-0621/
[project]
name = "pyadv-course"
dynamic = ["version"]
description = "Demo package for course."
readme = "README.md"
requires-python = ">=3.8"
license = { text = "BSD-3-Clause" }
authors = [{ name = "Guillaume Witz", email = "guillaume.witz@unibe.ch" }]
# https://pypi.org/classifiers/
classifiers = [
    "Development Status :: 3 - Alpha",
    "License :: OSI Approved :: BSD License",
    "Programming Language :: Python :: 3",
    "Programming Language :: Python :: 3.8",
    "Programming Language :: Python :: 3.9",
    "Programming Language :: Python :: 3.10",
    "Programming Language :: Python :: 3.11",
    "Programming Language :: Python :: 3.12",
    "Typing :: Typed",
```

pyproject.toml

Specifies the software
that builds to package.

Hatch should be the
future...

Versioning done
automatically via git tags
(more later)

Common metadata

Making plugin findable on
PyPi (e.g. napari plugin)

```
# https://peps.python.org/pep-0517/
[build-system]
requires = ["hatchling", "hatch-vcs"]
build-backend = "hatchling.build"

# https://hatch.pypa.io/latest/config/metadata/
[tool.hatch.version]
source = "vcs"

# read more about configuring hatch at:
# https://hatch.pypa.io/latest/config/build/
[tool.hatch.build.targets.wheel]
only-include = ["src"]
sources = ["src"]

# https://peps.python.org/pep-0621/
[project]
name = "pyadv-course"
dynamic = ["version"]
description = "Demo package for course."
readme = "README.md"
requires-python = ">=3.8"
license = { text = "BSD-3-Clause" }
authors = [{ name = "Guillaume Witz", email = "guillaume.witz@unibe.ch" }]
# https://pypi.org/classifiers/
classifiers = [
    "Development Status :: 3 - Alpha",
    "License :: OSI Approved :: BSD License",
    "Programming Language :: Python :: 3",
    "Programming Language :: Python :: 3.8",
    "Programming Language :: Python :: 3.9",
    "Programming Language :: Python :: 3.10",
    "Programming Language :: Python :: 3.11",
    "Programming Language :: Python :: 3.12",
    "Typing :: Typed",
]
```

pyproject.toml

Dependencies

Optional dependencies, e.g.
pip install aicsimageio[nd2]

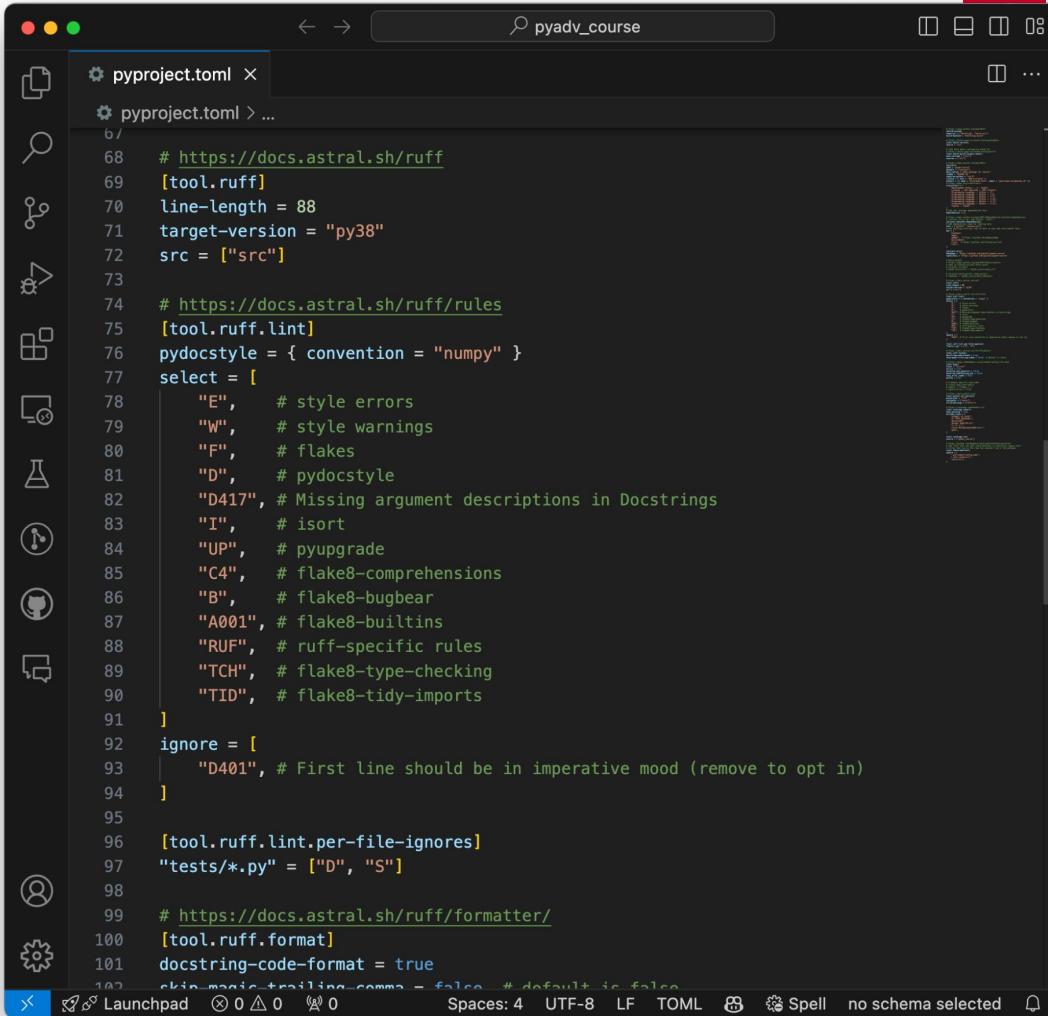
The screenshot shows a terminal window with the title bar "pyadv_course". The main content is a code editor displaying the `pyproject.toml` file. The file contains configuration sections like [project], [project.optional-dependencies], and [project.urls]. A red arrow points from the "Dependencies" text to the line `dependencies = []`. Another red arrow points from the "Optional dependencies" text to the section header `[project.optional-dependencies]`. The code editor has a dark theme with syntax highlighting for Python and TOML.

```
17 [project]
26 classifiers = [
31     "Programming Language :: Python :: 3.9",
32     "Programming Language :: Python :: 3.10",
33     "Programming Language :: Python :: 3.11",
34     "Programming Language :: Python :: 3.12",
35     "Typing :: Typed",
36 ]
37 # add your package dependencies here
38 dependencies = []
39
40 # https://peps.python.org/pep-0621/#dependencies-optional-dependencies
41 # "extras" (e.g. for `pip install .[test]`)
42 [project.optional-dependencies]
43 # add dependencies used for testing here
44 test = ["pytest", "pytest-cov"]
45 # add anything else you like to have in your dev environment here
46 dev = [
47     "ipython",
48     "mypy",
49     "pdbpp", # https://github.com/pdbpp/pdbpp
50     "pre-commit",
51     "rich", # https://github.com/Textualize/rich
52     "ruff",
53 ]
54
55 [project.urls]
56 homepage = "https://github.com/guiwitz/pyadv-course"
57 repository = "https://github.com/guiwitz/pyadv-course"
58
59 # Entry points
60 # https://peps.python.org/pep-0621/#entry-points
61 # same as console_scripts entry point
62 # [project.scripts]
63 # pyadv-course-cli = "pyadv_course:main_cli"
```

pyproject.toml

Additional sections for:

- formatting (ruff), absent for simpler package mode
- testing (pytest, coverage)
- code typing (mypy)



```
pyproject.toml x
pyproject.toml > ...
6/
68 # https://docs.astral.sh/ruff
69 [tool.ruff]
70 line-length = 88
71 target-version = "py38"
72 src = ["src"]
73
74 # https://docs.astral.sh/ruff/rules
75 [tool.ruff.lint]
76 pydocstyle = { convention = "numpy" }
77 select = [
78     "E",      # style errors
79     "W",      # style warnings
80     "F",      # flakes
81     "D",      # pydocstyle
82     "D417",   # Missing argument descriptions in Docstrings
83     "I",      # isort
84     "UP",     # pyupgrade
85     "C4",     # flake8-comprehensions
86     "B",      # flake8-bugbear
87     "A001",   # flake8-builtins
88     "RUF",    # ruff-specific rules
89     "TCH",    # flake8-type-checking
90     "TID",    # flake8-tidy-imports
91 ]
92 ignore = [
93     "D401",   # First line should be in imperative mood (remove to opt in)
94 ]
95
96 [tool.ruff.lint.per-file-ignores]
97 "tests/*.py" = ["D", "S"]
98
99 # https://docs.astral.sh/ruff/formatter/
100 [tool.ruff.format]
101 docstring-code-format = true
102 skip_magic_trailing_comma = false # default is false
```

The screenshot shows a terminal window titled "pyadv_course" displaying a "pyproject.toml" file. The file contains configuration for tools like Ruff, Pydocstyle, and Flake8. The code is color-coded: comments are grey, strings are green, and tool section names are yellow. The terminal interface includes standard macOS-style icons for file operations and a status bar at the bottom.

Push to GitHub

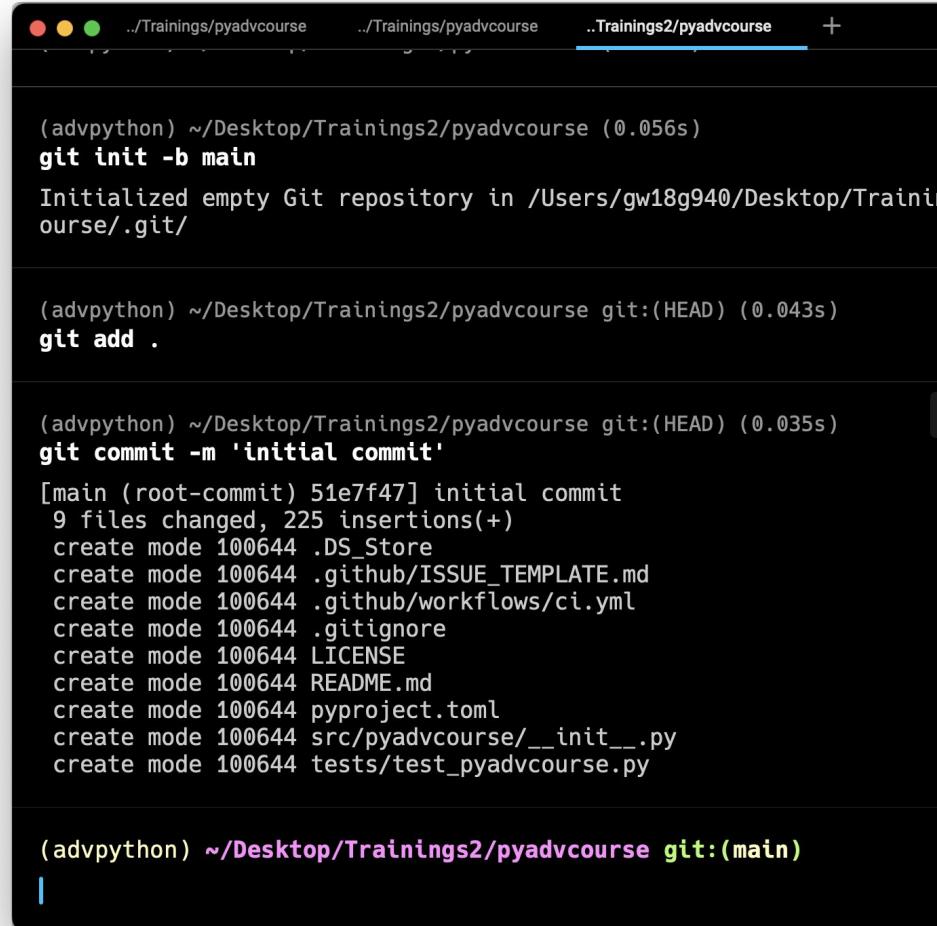
Initialize git

Eventually we want to push to GitHub
and use GitHub services

So we need to 1) initialize git, 2) connect
to GitHub

Inside your project folder use:

```
git init -b main
git add .
git commit -m 'initial commit'
```



The screenshot shows a macOS terminal window with three tabs open. The active tab is titled 'Trainings2/pyadvcourse'. The terminal output is as follows:

```
(advpython) ~/Desktop/Trainings2/pyadvcourse (0.056s)
git init -b main
Initialized empty Git repository in /Users/gw18g940/Desktop/Trainings2/pyadvcourse/.git/

(advpython) ~/Desktop/Trainings2/pyadvcourse git:(HEAD) (0.043s)
git add .

(advpython) ~/Desktop/Trainings2/pyadvcourse git:(HEAD) (0.035s)
git commit -m 'initial commit'
[main (root-commit) 51e7f47] initial commit
 9 files changed, 225 insertions(+)
  create mode 100644 .DS_Store
  create mode 100644 .github/ISSUE_TEMPLATE.md
  create mode 100644 .github/workflows/ci.yml
  create mode 100644 .gitignore
  create mode 100644 LICENSE
  create mode 100644 README.md
  create mode 100644 pyproject.toml
  create mode 100644 src/pyadvcourse/__init__.py
  create mode 100644 tests/test_pyadvcourse.py

(advpython) ~/Desktop/Trainings2/pyadvcourse git:(main)
```

Create a repository on Github

The screenshot shows a GitHub profile page for a user named Guillaume Witz (guiwitz). The page includes a profile picture, basic stats (39 followers, 0 following, 3 stars), a GitHub URL (guiwitz.github.io), and a 'Highlights' section mentioning Arctic Code Vault Contributor status and being a PRO member.

The main focus is the 'Repositories' section, which is highlighted with a red circle. This section shows a list of repositories:

- hpc_cloud**: Demo repo for course. Last updated 1 hour ago. Status: Jupyter Notebook.
- MorphoDynamics** [Private]: A Python package for tracking the deformation of biological cells. Last updated 2 days ago. Status: Python.
- Pythonaction** [Private]: Repos to test github-actions. Last updated 11 days ago. Status: Python.
- jupyter-omeroanalysis-desktop**: Forked from manics/jupyter-omeroanalysis-desktop. Status: PRO.

A green 'New' button is also highlighted with a red circle, indicating where a new repository can be created.

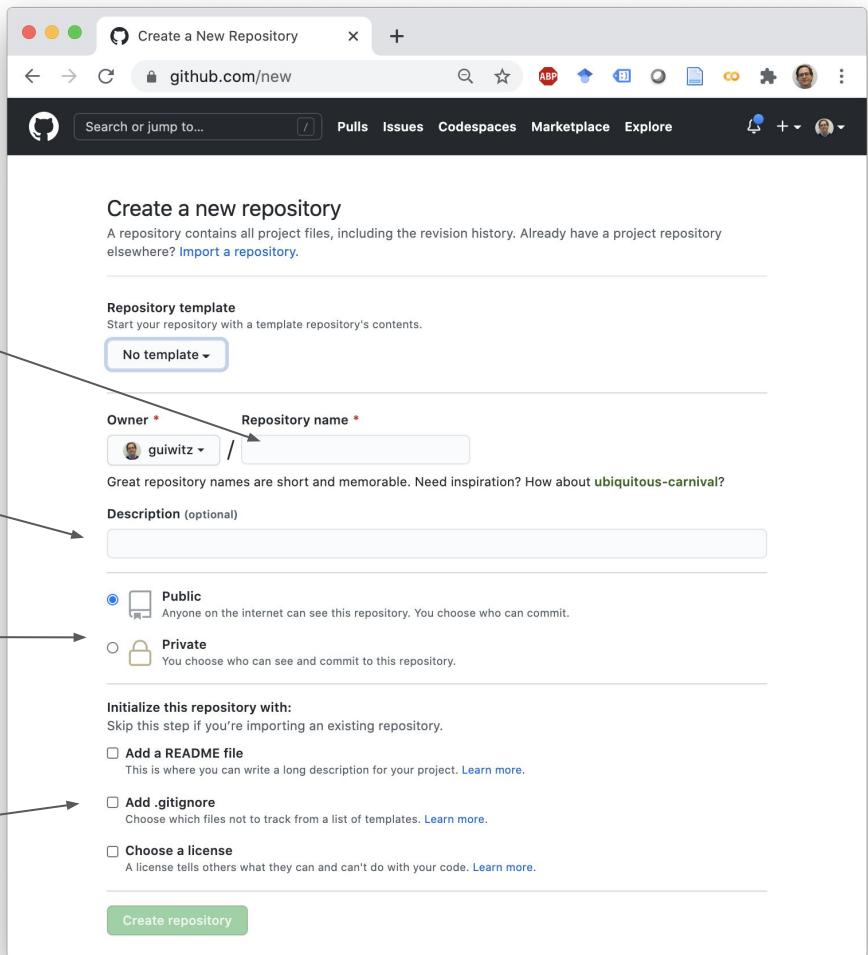
Create a repository

Choose the same name as your project: pyadv-course

Add a short description

Make your repository public. You can keep not-yet released projects (e.g. article) private

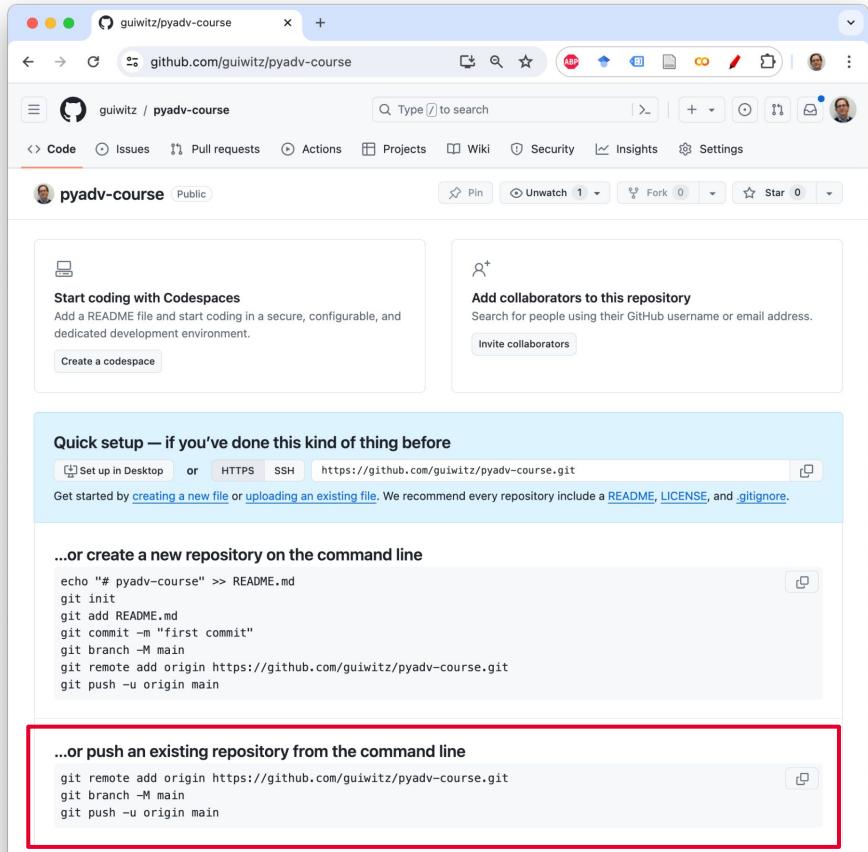
Add NO files



Create a repository

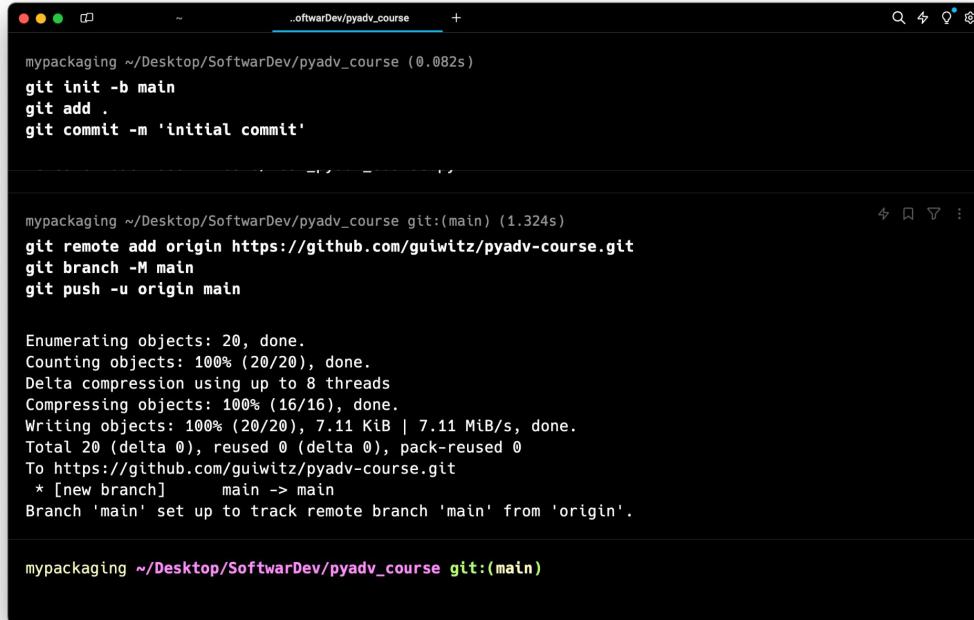
Connect your local repository to the GitHub repository

You will be prompted to login to GitHub in one way or another...



Connect your local project to GitHub

```
git remote add origin https://github.com/guiwitz/pyadv-course.git  
git branch -M main  
git push -u origin main
```



The screenshot shows a macOS terminal window with a dark theme. The title bar reads ".SoftwarDev/pyadv_course". The terminal displays the following command sequence and its output:

```
mypackaging ~/Desktop/SoftwarDev/pyadv_course (0.082s)  
git init -b main  
git add .  
git commit -m 'initial commit'  
  
mypackaging ~/Desktop/SoftwarDev/pyadv_course git:(main) (1.324s)  
git remote add origin https://github.com/guiwitz/pyadv-course.git  
git branch -M main  
git push -u origin main  
  
Enumerating objects: 20, done.  
Counting objects: 100% (20/20), done.  
Delta compression using up to 8 threads  
Compressing objects: 100% (16/16), done.  
Writing objects: 100% (20/20), 7.11 KiB | 7.11 MiB/s, done.  
Total 20 (delta 0), reused 0 (delta 0), pack-reused 0  
To https://github.com/guiwitz/pyadv-course.git  
 * [new branch]      main -> main  
Branch 'main' set up to track remote branch 'main' from 'origin'.  
  
mypackaging ~/Desktop/SoftwarDev/pyadv_course git:(main)
```

Install project

Editable install

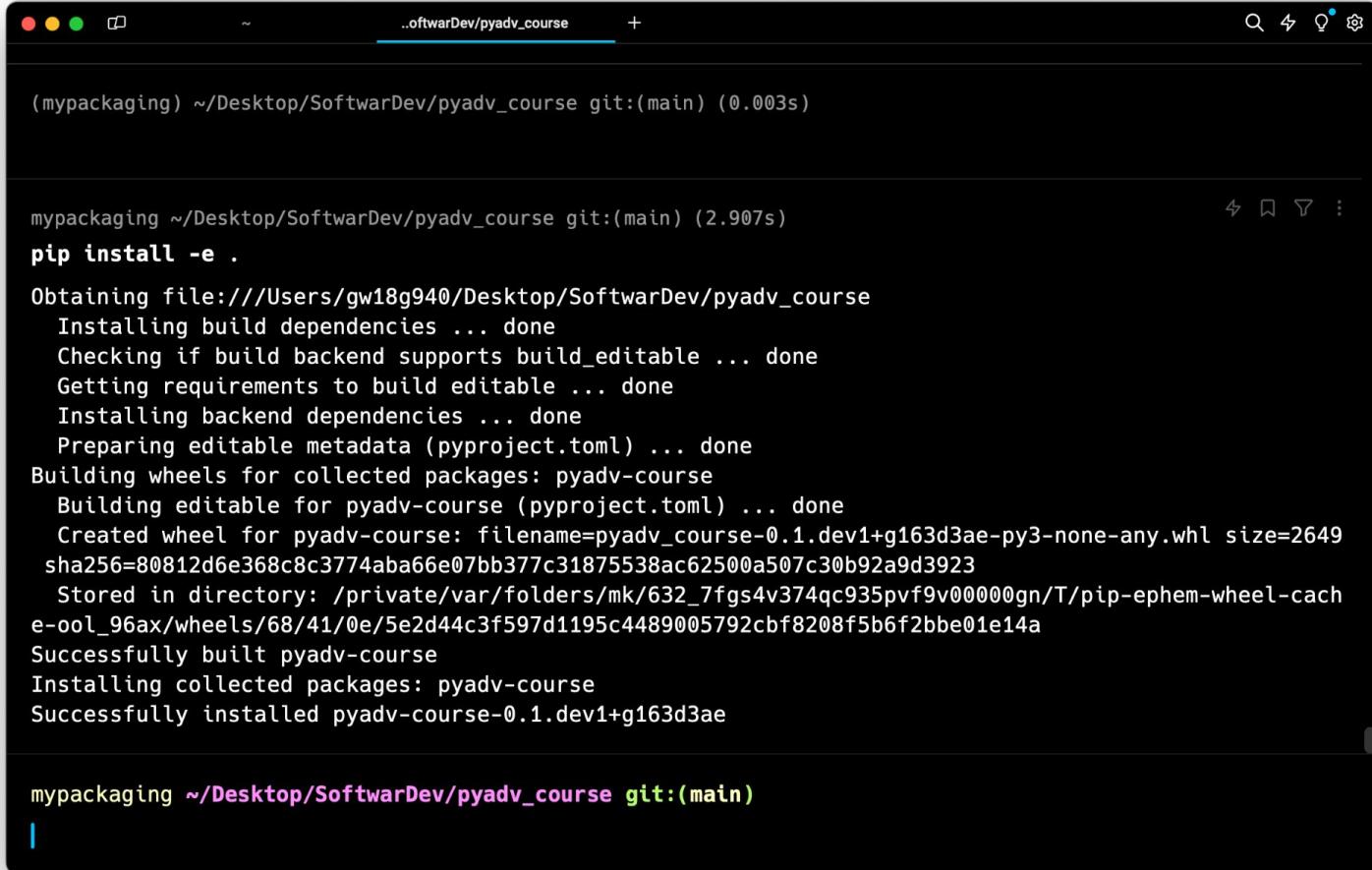
When you install a package with `pip install mypackage`, the code is put in a place where it should not be edited.

When you work on a project you want to be able to edit it.

You can install in editable mode i.e. the package you use (e.g. `import mypackage`) is the one that you can edit.

For this, cd to your package (level of `pyproject.toml`) and use (don't forget the point): **pip install -e .**

Editable install



The screenshot shows a macOS terminal window with a dark theme. The title bar reads ".oftwarDev/pyadv_course". The terminal output is as follows:

```
(mypackaging) ~/Desktop/SoftwarDev/pyadv_course git:(main) (0.003s)

mypackaging ~/Desktop/SoftwarDev/pyadv_course git:(main) (2.907s)
pip install -e .

Obtaining file:///Users/gw18g940/Desktop/SoftwarDev/pyadv_course
  Installing build dependencies ... done
  Checking if build backend supports build_editable ... done
  Getting requirements to build editable ... done
  Installing backend dependencies ... done
  Preparing editable metadata (pyproject.toml) ... done
Building wheels for collected packages: pyadv-course
  Building editable for pyadv-course (pyproject.toml) ... done
  Created wheel for pyadv-course: filename=pyadv_course-0.1.dev1+g163d3ae-py3-none-any.whl size=2649
sha256=80812d6e368c8c3774aba66e07bb377c31875538ac62500a507c30b92a9d3923
  Stored in directory: /private/var/folders/mk/632_7fgs4v374qc935pvf9v00000gn/T/pip-ephem-wheel-cach
e-ool_96ax/wheels/68/41/0e/5e2d44c3f597d1195c4489005792cbf8208f5b6f2bbe01e14a
Successfully built pyadv-course
Installing collected packages: pyadv-course
Successfully installed pyadv-course-0.1.dev1+g163d3ae

mypackaging ~/Desktop/SoftwarDev/pyadv_course git:(main)
```

Use your package

Start Jupyter:

```
jupyter lab
```

Create a notebook and try to import
your package.

You should NOT get an error but...
the package is empty

It has a version though!

```
[1]: import pyadv_course
```

```
[2]: pyadv_course.__version__
```

```
[2]: '0.1.dev1+g163d3ae'
```

Edit your package

Open the package in VSCode

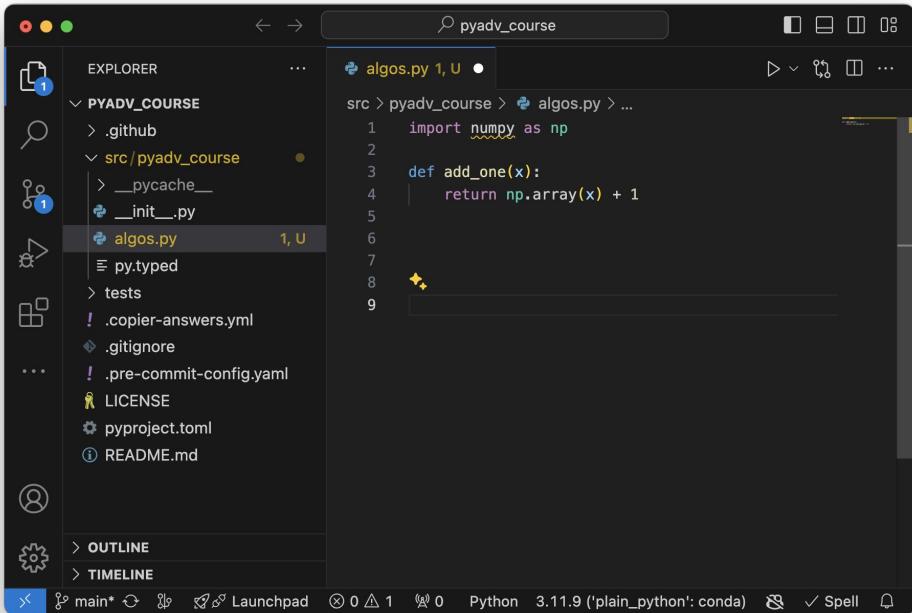
Open Folder -> pyadv_course

Let's add an `algos.py` file inside `src`

And let's add some dummy code:

```
import numpy as np
```

```
def add_one(x):  
    return np.array(x) + 1
```



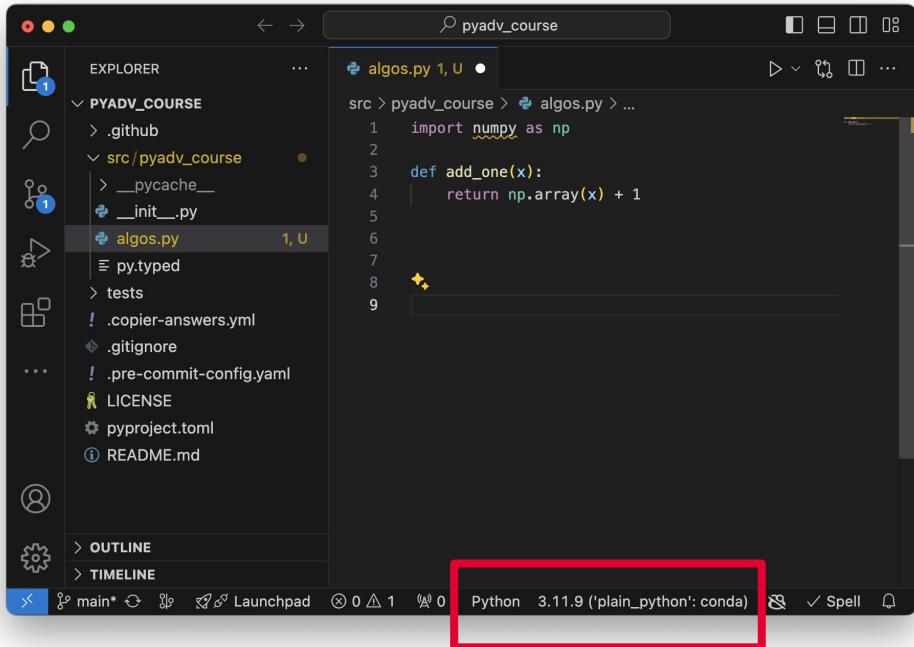
Open the package in VSCode

Let's fix the environment

And commit the file:

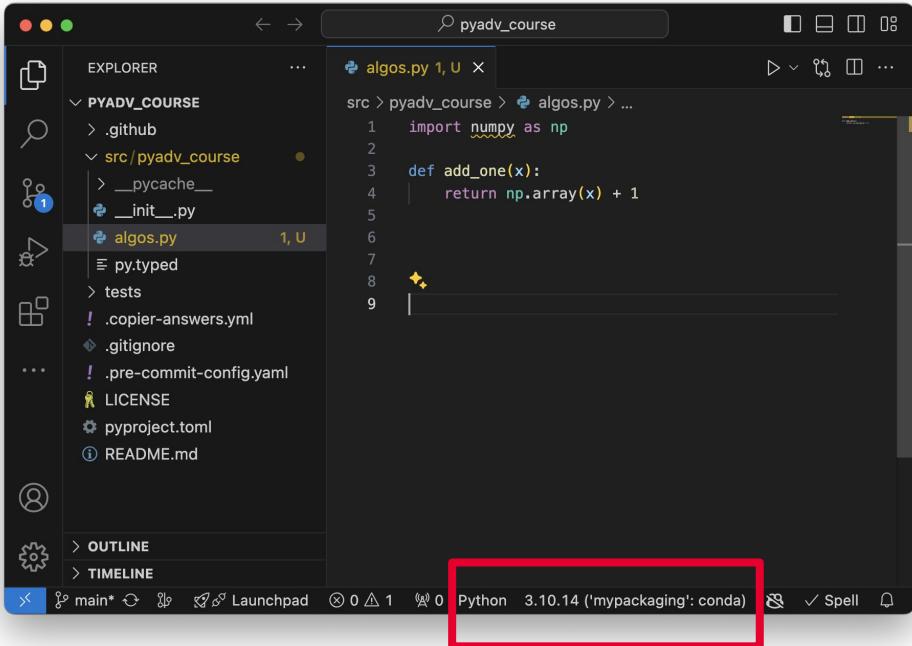
```
git add src/algos.py
```

```
git commit -m 'adding algos'
```



Open the package in VSCode

Let's fix the environment



Use your package

Try now to import the function

Module exists but dependency is missing!

```
[3]: from pyadv_course import algos
-----
-----
ModuleNotFoundError                                     Traceback (most
recent call last)
Cell In[3], line 1
----> 1 from pyadv_course import algos

File ~/Desktop/SoftwareDev/pyadv_course/src/pyadv_course/al
gos.py:1
----> 1 import numpy as np
      3 def add_one(x):
      4     return np.array(x) + 1

ModuleNotFoundError: No module named 'numpy'
```

Dependencies

Adding dependencies

No dependencies specified.

We can:

- install on the fly: cumbersome
- specify dependencies in the toml file -> dependency will always be available when installing package

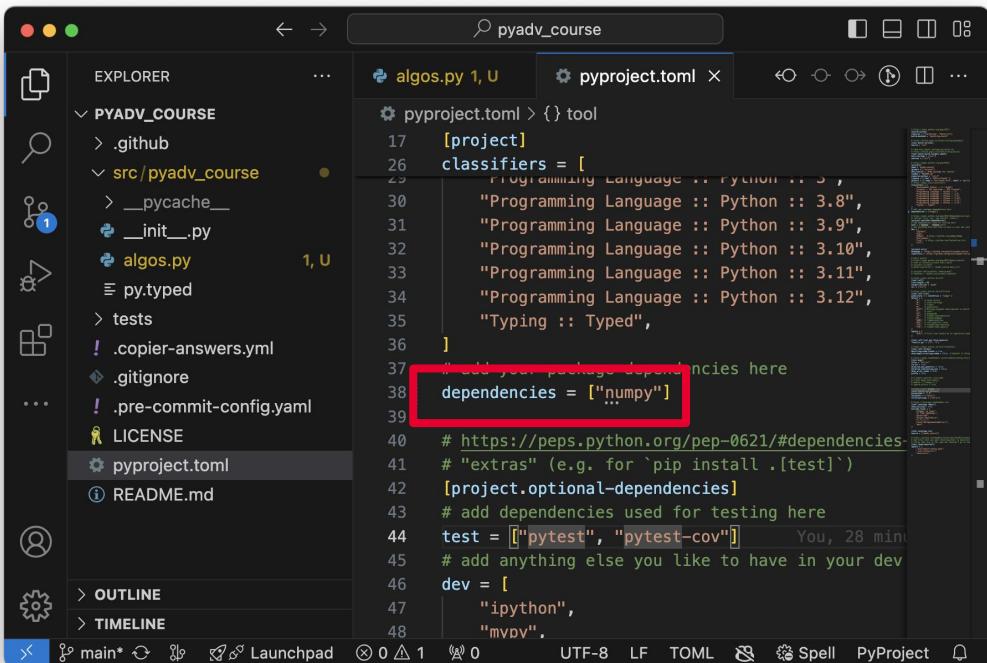
A screenshot of a code editor (VS Code) showing a project structure in the Explorer sidebar and a code editor window. The code editor displays the `pyproject.toml` file for a project named `PYADV_COURSE`. The file contains configuration for Python packages, including classifiers for various Python versions and a section for dependencies. A red box highlights the line `dependencies = []`, which is currently empty. The code editor interface includes tabs for other files like `algos.py` and `pyproject.toml`, and various status icons at the bottom.

```
[project]
classifiers = [
    "Programming Language :: Python :: 3",
    "Programming Language :: Python :: 3.8",
    "Programming Language :: Python :: 3.9",
    "Programming Language :: Python :: 3.10",
    "Programming Language :: Python :: 3.11",
    "Programming Language :: Python :: 3.12",
    "Typing :: Typed",
]
# add your package dependencies here
dependencies = []

# https://peps.python.org/pep-0621/#dependencies
# "extras" (e.g. for `pip install .[test]`)
[project.optional-dependencies]
# add dependencies used for testing here
test = ["pytest", "pytest-cov"]
# add anything else you like to have in your dev
dev = [
    "ipython",
    "mypy",
]
```

Adding dependencies

- After adding the dependency, we re-install the package:
- pip install -e .**
- This triggers installation of dependencies!
- Dependencies are installed from PyPi



A screenshot of a code editor showing a `pyproject.toml` file. The file contains configuration for a Python project named `PYADV_COURSE`. The `dependencies` section is highlighted with a red rectangle and contains the entry `"numpy"`.

```
pyproject.toml
[project]
classifiers = [
    "Programming Language :: Python :: 3",
    "Programming Language :: Python :: 3.8",
    "Programming Language :: Python :: 3.9",
    "Programming Language :: Python :: 3.10",
    "Programming Language :: Python :: 3.11",
    "Programming Language :: Python :: 3.12",
    "Typing :: Typed",
]
# Add your package dependencies here
dependencies = ["numpy"]
```

Adding dependencies

- After adding the dependency, we re-install the package:
- pip install -e .**
- This triggers installation of dependencies!
- Dependencies are installed from PyPi

```
mypackaging ~/Desktop/SoftwarDev/pyadv_course git:(main) (6.648s)
pip install -e .
  Installing backend dependencies ... done
  Preparing editable metadata (pyproject.toml) ... done
Collecting numpy (from pyadv-course==0.1.dev1+g163d3ae.d20240605)
  Downloading numpy-1.26.4-cp310-cp310-macosx_11_0_arm64.whl.metadata (61 kB)
                                             61.1/61.1 kB 1.0 MB/s eta 0:00:00
  Downloading numpy-1.26.4-cp310-cp310-macosx_11_0_arm64.whl (14.0 MB)
                                             14.0/14.0 MB 14.2 MB/s eta 0:00:00
Building wheels for collected packages: pyadv-course
  Building editable for pyadv-course (pyproject.toml) ... done
  Created wheel for pyadv-course: filename=pyadv_course-0.1.dev1+g163d3ae.d20240605-py3-none-any.whl
size=2752 sha256=2111a465cbe4778908f8fe3310ab8de8dcbb3d4ba6c3c4c4b7372ef50fbff05d1
  Stored in directory: /private/var/folders/mk/632_7fgs4v374qc935pvf9v0000gn/T/pip-ephem-wheel-cache-qfm09n81/wheels/68/41/0e/5e2d44c3f597d1195c4489005792cbf8208f5b6f2bbe01e14a
Successfully built pyadv-course
Installing collected packages: numpy, pyadv-course
  Attempting uninstall: pyadv-course
    Found existing installation: pyadv-course 0.1.dev1+g163d3ae
    Uninstalling pyadv-course-0.1.dev1+g163d3ae:
      Successfully uninstalled pyadv-course-0.1.dev1+g163d3ae
Successfully installed numpy-1.26.4 pyadv-course-0.1.dev1+g163d3ae.d20240605

mypackaging ~/Desktop/SoftwarDev/pyadv_course git:(main)±
```

Adding optional dependencies

- On the model of test and dev, add you own name for optional dependencies, typically for functionalities used by only a fraction of users.

```
pyproject.toml
[project]
classifiers = [
    "Programming Language :: Python :: 3",
    "Programming Language :: Python :: 3.8",
    "Programming Language :: Python :: 3.9",
    "Programming Language :: Python :: 3.10",
    "Programming Language :: Python :: 3.11",
    "Programming Language :: Python :: 3.12",
    "Typing :: Typed",
]
# add your package dependencies here
dependencies = ["numpy"]

[project.optional-dependencies]
# add dependencies used for testing here
test = ["pytest", "pytest-cov"]
# add anything else you like to have in your dev
dev = [
    "ipython",
    "mypy",
]
```

Adding dependencies

Our import now works!

```
[4]: from pyadv_course import algos
```

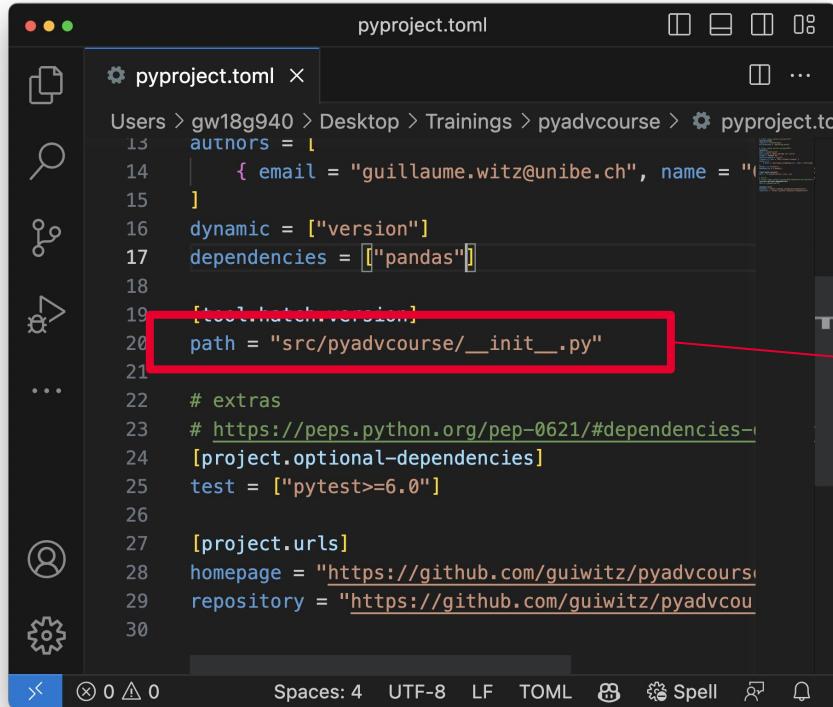
Hot reloading

When modifying a package, one has to restart the notebook's kernel and import it for changes to take effect. To avoid this, one can add this before any import statement to be able to update packages "live":

```
%load_ext autoreload  
%autoreload 2
```

Versions

Hard-coded version



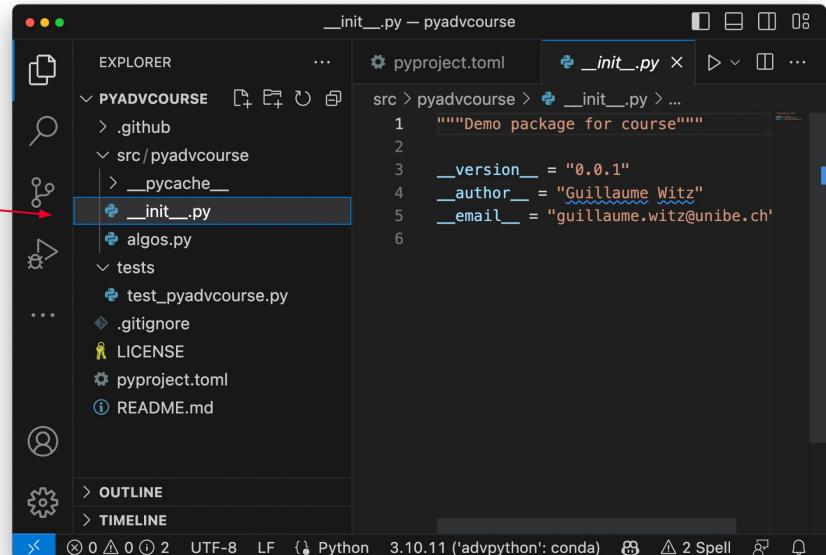
```
pyproject.toml
```

```
pyproject.toml
Users > gw18g940 > Desktop > Trainings > pyadvcourse > pyproject.toml
13 authors = [
14     { email = "guillaume.witz@unibe.ch", name = "Guillaume Witz" }
15 ]
16 dynamic = ["version"]
17 dependencies = [ "pandas" ]
18
19 [tool.hatch.version]
20 path = "src/pyadvcourse/__init__.py"
21
22 # extras
23 # https://peps.python.org/pep-0621/#dependencies-and-extras
24 [project.optional-dependencies]
25 test = ["pytest>=6.0"]
26
27 [project.urls]
28 homepage = "https://github.com/guiwitz/pyadvcourse"
29 repository = "https://github.com/guiwitz/pyadvcou
```

Spaces: 4 UTF-8 LF TOML Spell

```
[7]: import pyadvcourse
pyadvcourse.__version__
```

```
[7]: '0.0.1'
```



```
__init__.py -- pyadvcourse
```

```
EXPLORER
PYADVCOURSE
> .github
< src / pyadvcourse
    > __pycache__
        __init__.py
    algos.py
    tests
        test_pyadvcourse.py
    .gitignore
    LICENSE
    pyproject.toml
    README.md
```

```
src > pyadvcourse > __init__.py > ...
1 """Demo package for course"""
2
3 __version__ = "0.0.1"
4 __author__ = "Guillaume Witz"
5 __email__ = "guillaume.witz@unibe.ch"
```

Spaces: 0 Δ 0 1 2 UTF-8 LF Python 3.10.11 ('advpython': conda) Spell

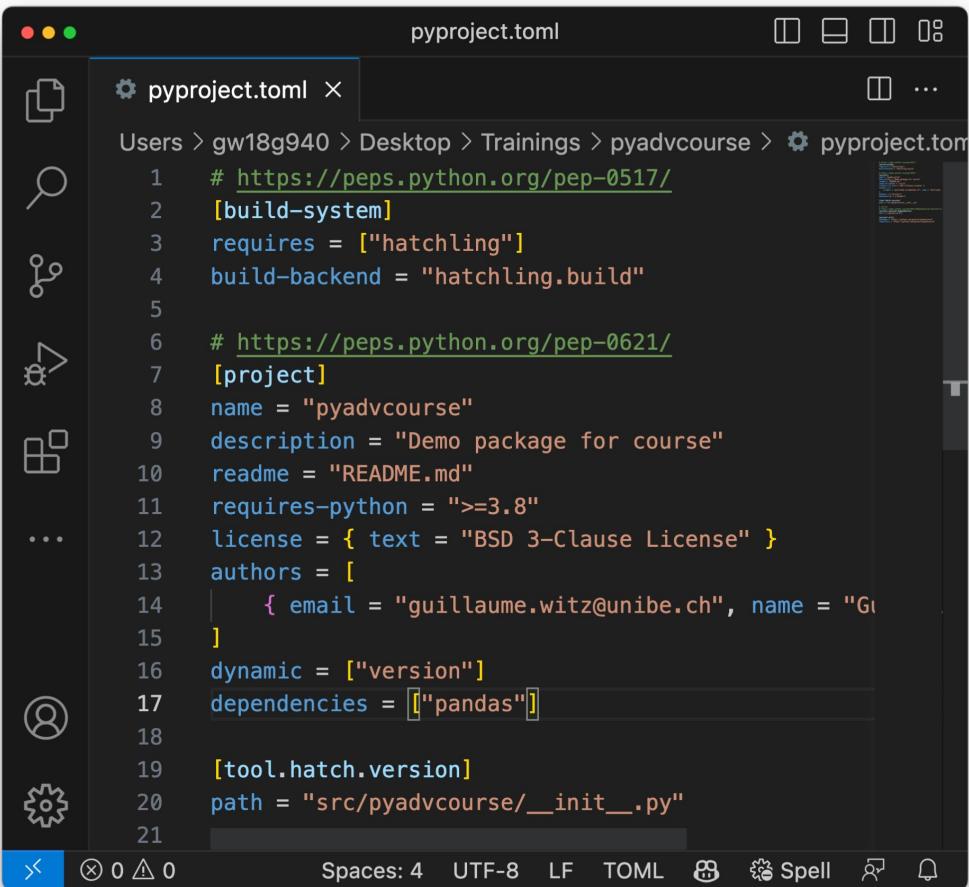
Dynamic version

Hard-coded versions are difficult to "maintain"

A better way is to dynamically update the version with changes

One solution: use git

For this we tell hatch to use git versioning to define the version



The screenshot shows a code editor window with a dark theme. The title bar says "pyproject.toml". The left sidebar has icons for file, search, file operations, and more. The main area contains the following code:

```
pyproject.toml
-----
1 # https://peps.python.org/pep-0517/
2 [build-system]
3 requires = ["hatchling"]
4 build-backend = "hatchling.build"
5
6 # https://peps.python.org/pep-0621/
7 [project]
8 name = "pyadvcourse"
9 description = "Demo package for course"
10 readme = "README.md"
11 requires-python = ">=3.8"
12 license = { text = "BSD 3-Clause License" }
13 authors = [
14     { email = "guillaume.witz@unibe.ch", name = "Guillaume Witz" }
15 ]
16 dynamic = ["version"]
17 dependencies = ["pandas"]
18
19 [tool.hatch.version]
20 path = "src/pyadvcourse/__init__.py"
```

At the bottom, there are status indicators: "Spaces: 4", "UTF-8", "LF", "TOML", and icons for spell check and other tools.

Dynamic version

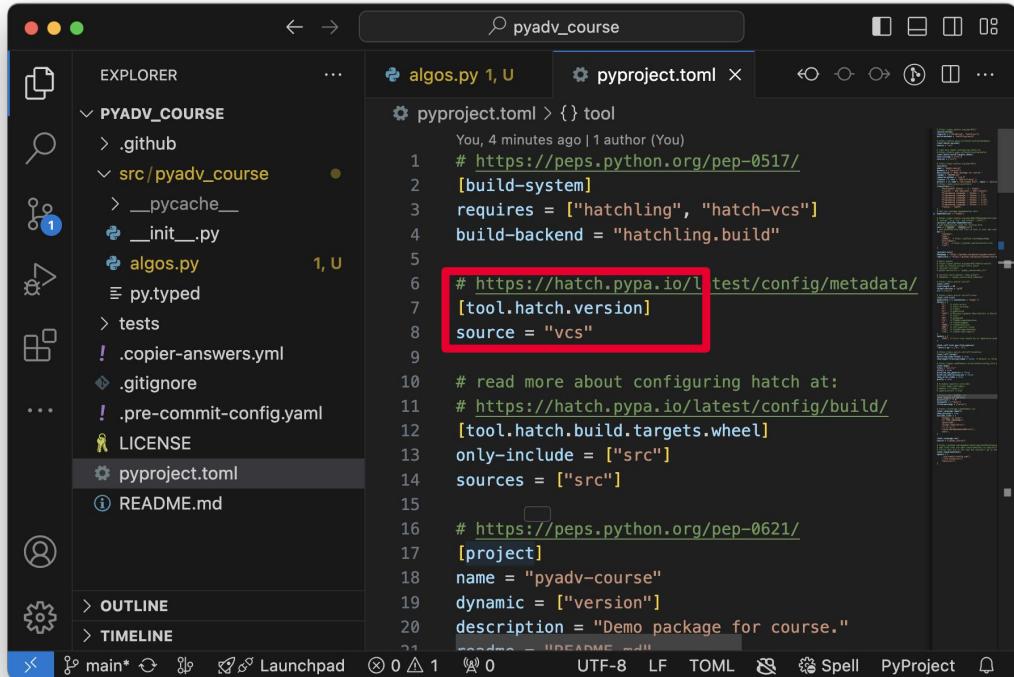
Hard-coded versions are difficult to "maintain"

A better way is to dynamically update the version with changes

One solution: use git

For this we tell hatch to use git versioning to define the version

From the copier, template, the project already uses VCS for versioning



```
pyproject.toml > {} tool
You, 4 minutes ago | 1 author (You)
1 # https://peps.python.org/pep-0517/
2 [build-system]
3 requires = ["hatchling", "hatch-vcs"]
4 build-backend = "hatchling.build"
5
6 # https://hatch.pypa.io/latest/config/metadata/
7 [tool.hatch.version]
8 source = "vcs"
9
10 # read more about configuring hatch at:
11 # https://hatch.pypa.io/latest/config/build/
12 [tool.hatch.build.targets.wheel]
13 only-include = ["src"]
14 sources = ["src"]
15
16 # https://peps.python.org/pep-0621/
17 [project]
18 name = "pyadv-course"
19 dynamic = ["version"]
20 description = "Demo package for course."
21 encoding = "UTF-8" LF TOML & Spell PyProject
```

Tagging the version

We add a git tag to the current version (comment about the status) and locally re-install:

```
git tag -a v0.1.1 -m "v0.1.1"  
pip install -e .
```

```
[1]: import pyadv_course
```

```
[2]: pyadv_course.__version__
```

```
[2]: '0.1.2.dev0+g7c53523.d20240605'
```

Tags

The image shows two screenshots of the GitHub interface. The left screenshot displays the main repository page for 'guiwitz/pyadvcourse'. A red box highlights the '2 tags' button in the top navigation bar. The right screenshot shows the 'Tags' tab within the 'Releases' section of the same repository, listing two tags: 'v0.1.1' and 'v0.1.0'.

Left Screenshot (Repository Page):

- Repository: `guiwitz/pyadvcourse` (Public)
- Code tab selected
- Branch: `main`
- Commits: 2 commits (15480ea, 4 minutes ago)
- Tags: 2 tags (highlighted by a red box)
- Actions: Go to file, Add file, Code dropdown

Right Screenshot (Tags Tab):

- Repository: `guiwitz/pyadvcourse` (Public)
- Tags tab selected
- Tags listed:
 - v0.1.1 (4 minutes ago, 15480ea, zip, tar.gz)
 - v0.1.0 (5 minutes ago, 15480ea, zip, tar.gz)