

# Python packaging, Part 2

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



Testing: what does my code do when  
I'm not looking at it 

# Why testing

1. You want to be sure your code is doing what you think: give it examples where you know the answer and check
2. You modify your code and want to make sure the behavior doesn't change
3. Investigate edge cases: helps you add warnings, errors

# Testing a function

You designed a function e.g.

sum\_thresholding\_fun

The function rounds a list of numbers and counts elements larger than a threshold

Does it really do that?

With

[0.1, 0.4, 0.0.3, 0.9, 1.1, 1.2]

with a threshold of 0 we should get 3

```
import numpy as np

def sum_thresholding_fun(x, threshold):
    """Threshold array and sum positive elemens"""

    x = np.round(x)
    x_th = x > threshold
    out = np.sum(x_th)

    return out

my_array = np.array([0.1, 0.4, 0.3, 0.9, 1.1, 1.2])

out = sum_thresholding_fun(my_array, 0)

out
```

# Testing a function

Three months later, we make a small modification in the code: we replace round with floor.

Does it affect the result of the code?

Yes! The result is 2.

How to test that?

```
import numpy as np

def sum_thresholding_fun(x, threshold):
    """Threshold array and sum positive elemens"""

    x = np.floor(x)
    x_th = x > threshold
    out = np.sum(x_th)

    return out

my_array = np.array([0.1, 0.4, 0.3, 0.9, 1.1, 1.2])
out = sum_thresholding_fun(my_array, 0)

out
```

# Natural solution: if to test solution

```
my_array = np.array([0.1, 0.4, 0.3, 0.9, 1.1, 1.2])  
  
out = sum_thresholding_fun(my_array, 0)  
if out != 3:  
    raise ValueError("sum_thresholding_fun failed")
```

```
-----  
ValueError                                                 Traceback (most  
recent call last)  
Cell In[41], line 5  
      3 out = sum_thresholding_fun(my_array, 0)  
      4 if out != 3:  
----> 5       raise ValueError("sum_thresholding_fun faile  
d")  
  
ValueError: sum_thresholding_fun failed
```

# Simpler alternative: assert

The built-in function `assert` checks the validity of an assertion.

Works with any assertion

Raises a specific assertion error

```
[47]: a = 5
```

```
assert a == 5
```

```
[48]: assert a == 6
```

```
AssertionError  
Cell In[48], line 1  
----> 1 assert a == 6
```

Traceback

```
[50]: a = 5  
mylist = [5, 4, 2]
```

```
assert a in mylist
```

```
[51]: a = 6  
assert a in mylist
```

```
AssertionError  
Cell In[51], line 2  
1 a = 6  
----> 2 assert a in mylist
```

Traceback

```
AssertionError:
```

# Simpler alternative: assert

The built-in function `assert` checks the validity of an assertion.

Works with any assertion

Raises a specific assertion error

```
[54]: a = 'my text'  
  
      assert isinstance(a, str)
```

```
[55]: a = 5  
  
      assert isinstance(a, str)
```

```
-----  
AssertionError  
recent call last)  
Cell In[55], line 3  
  1 a = 5  
----> 3 assert isinstance(a, str)  
  
AssertionError:
```

Traceback (

# Simpler alternative: assert

The built-in function `assert` checks the validity of an assertion.

Works with any assertion

Raises a specific assertion error

For our function

```
[59]: import numpy as np

def sum_thresholding_fun(x, threshold):
    """Threshold array and sum positive elemens"""

    x = np.floor(x)
    x_th = x > threshold
    out = np.sum(x_th)

    return out

my_array = np.array([0.1, 0.4, 0.3, 0.9, 1.1, 1.2])

out = sum_thresholding_fun(my_array, 0)

assert out == 3
```

```
-----
AssertionError                                         Traceback (most
recent call last)
Cell In[59], line 16
  12 my_array = np.array([0.1, 0.4, 0.3, 0.9, 1.1, 1.
  2])
  14 out = sum_thresholding_fun(my_array, 0)
--> 16 assert out == 3

AssertionError:
```

# More specific assert

We can add a message to assert to make it clear what's wrong

```
assert out == 3, 'Thresholding not working'
```

```
AssertionError  
recent call last)  
Cell In[61], line 1  
----> 1 assert out == 3, 'Thresholding not working'
```

Traceback (most

```
AssertionError: Thresholding not working
```

# More specific assert

We can add a message to assert to make it clear what's wrong

We can even include the actual and expected result in out description using an f-string

```
assert out == 3, f'thresholding should give 3 but is {out}'
```

```
AssertionError  
ent call last)  
Cell In[64], line 1  
----> 1 assert out == 3, f'thresholding should give 3 but is  
{out}'
```

Traceback (most recent call last)

```
AssertionError: thresholding should give 3 but is 2
```

# Comparing arrays

More complex objects can't just be compared with ==

E.g for Numpy arrays, what exactly does myarray1 == myarray2 mean?

Numpy provides specific assert functions for this purpose

Provides a detailed comparison

```
[68]: myarray1 = np.array([1,2,3])
myarray2 = np.array([1,3,2])

np.testing.assert_array_equal(myarray1, myarray2)
```

---

```
AssertionError                                         Traceback (most recent call last)
Cell In[68], line 4
      1 myarray1 = np.array([1,2,3])
      2 myarray2 = np.array([1,3,2])
----> 4 np.testing.assert_array_equal(myarray1, myarray2)

[... skipping hidden 1 frame]

File ~/mambaforge/envs/advpython/lib/python3.10/contextlib.py:79, in ContextDecorator._call_.<locals>.inner(*args, **kwds)
    76     @wraps(func)
    77     def inner(*args, **kwds):
    78         with self._recreate_cm():
----> 79             return func(*args, **kwds)

File ~/mambaforge/envs/advpython/lib/python3.10/site-packages/numpy/testing/_private/utils.py:862, in assert_array_compare(comparison, x, y, err_msg, verbose, header, precision, equal_nan, equal_inf, strict)
    858         err_msg += '\n' + '\n'.join(remarks)
    859         msg = build_err_msg([ox, oy], err_msg,
    860                             verbose=verbose, header=header,
    861                             names=('x', 'y'), precision=precision)
----> 862         raise AssertionError(msg)
    863 except ValueError:
    864     import traceback

AssertionError:
Arrays are not equal

Mismatched elements: 2 / 3 (66.7%)
Max absolute difference: 1
Max relative difference: 0.5
x: array([1, 2, 3])
y: array([1, 3, 2])
```

# Comparing arrays

More complex objects can't just be compared with ==

E.g for Numpy arrays, what exactly does `myarray1 == myarray2` mean?

Numpy provides specific assert functions for this purpose

Provides a detailed comparison

```
[68]: myarray1 = np.array([1,2,3])
myarray2 = np.array([1,3,2])

np.testing.assert_array_equal(myarray1, myarray2)

AssertionError                                     Traceback (most recent call last)
Cell In[68], line 4
      1 myarray1 = np.array([1,2,3])
      2 myarray2 = np.array([1,3,2])
----> 4 np.testing.assert_array_equal(myarray1, myarray2)

[... skipping hidden 1 frame]

File ~/mambaforge/envs/advpython/lib/python3.10/contextlib.py:79, in ContextDecorator._call_.<locals>.inner(*args, **kwds)
    76     @wraps(func)
    77     def inner(*args, **kwds):
    78         with self._recreate_cm():
----> 79             return func(*args, **kwds)

File ~/mambaforge/envs/advpython/lib/python3.10/site-packages/numpy/testing/_private/utils.py:862, in assert_array_compare(comparison, x, y, err_msg, verbose, header, precision, equal_nan, equal_inf, strict)
    858         err_msg += '\n' + '\n'.join(remarks)
    859         msg = build_err_msg([ox, oy], err_msg,
    860                             verbose=verbose, header=header,
    861                             names=(x, y), precision=precision)
----> 862         raise AssertionError(msg)
    863 except ValueError:
    864     import traceback

AssertionError:
Arrays are not equal

Mismatched elements: 2 / 3 (66.7%)
Max absolute difference: 1
Max relative difference: 0.5
x: array([1, 2, 3])
y: array([1, 3, 2])
```

# Comparing arrays: approx equal

```
[72]: myarray1 = np.array([1,2,3])
myarray2 = np.array([1,2,3]) + 0.01

[73]: myarray1
[73]: array([1, 2, 3])

[74]: myarray2
[74]: array([1.01, 2.01, 3.01])
```

**AssertionError:**  
 Arrays are not almost equal to 3 decimals  
 Mismatched elements: 3 / 3 (100%)  
 Max absolute difference: 0.01  
 Max relative difference: 0.00990099  
 x: array([1, 2, 3])  
 y: array([1.01, 2.01, 3.01])

```
: np.testing.assert_almost_equal(myarray1, myarray2, decimal=1)

: np.testing.assert_almost_equal(myarray1, myarray2, decimal=3)
-----
AssertionError                                         Traceback (most recent call last)
Cell In[77], line 1
----> 1 np.testing.assert_almost_equal(myarray1, myarray2, decimal=3)

File ~/mambaforge/envs/advpython/lib/python3.10/contextlib.py:79, in ContextDecorator._call_.<locals>.inner(*args, **kwds)
    76     @wraps(func)
    77     def inner(*args, **kwds):
    78         with self._recreate_cm():
--> 79             return func(*args, **kwds)

[... skipping hidden 1 frame]

File ~/mambaforge/envs/advpython/lib/python3.10/contextlib.py:79, in ContextDecorator._call_.<locals>.inner(*args, **kwds)
    76     @wraps(func)
    77     def inner(*args, **kwds):
    78         with self._recreate_cm():
--> 79             return func(*args, **kwds)

[... skipping hidden 1 frame]

File ~/mambaforge/envs/advpython/lib/python3.10/contextlib.py:79, in ContextDecorator._call_.<locals>.inner(*args, **kwds)
    76     @wraps(func)
    77     def inner(*args, **kwds):
    78         with self._recreate_cm():
--> 79             return func(*args, **kwds)
```

# Writing tests: automation

We can turn each test we write for a function into a function itself

Then we group all tests in a module

This will allow us to run all test in one go later on

```
import numpy as np

def sum_thresholding_fun(x, threshold):
    """Threshold array and sum positive elemens"""

    x = np.floor(x)
    x_th = x > threshold
    out = np.sum(x_th)

    return out

def test_sum_thresholding_fun():

    my_array = np.array([0.1, 0.4, 0.3, 0.9, 1.1, 1.2])

    out = sum_thresholding_fun(my_array, 0)

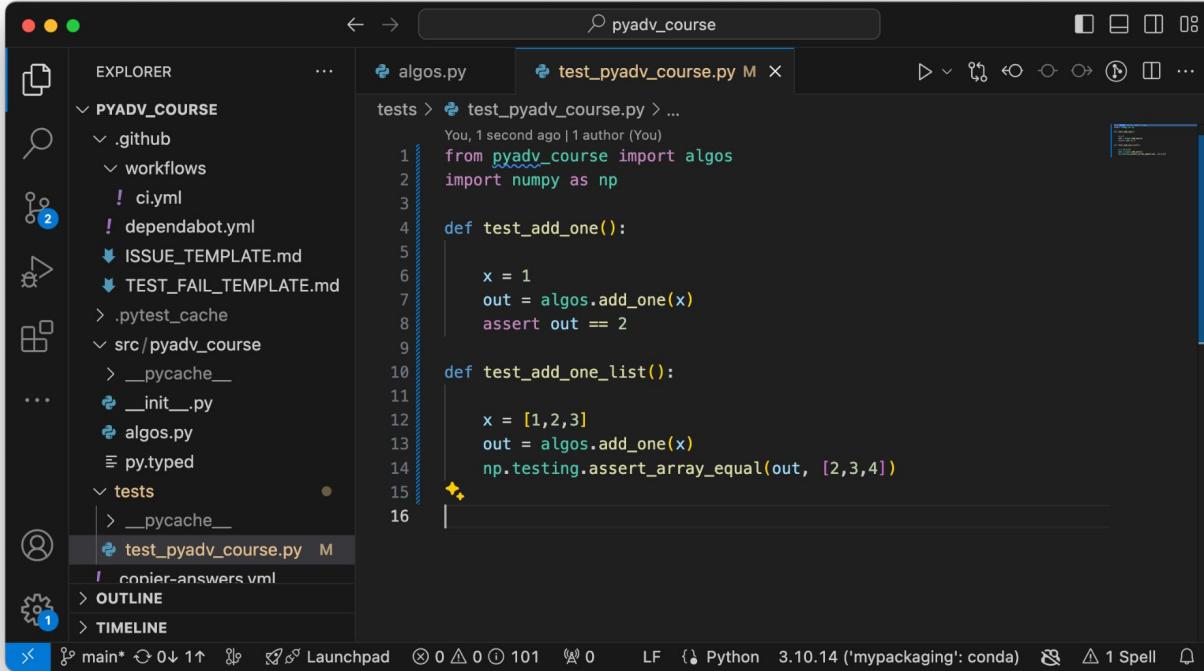
    assert out == 3
```

Tests for package, pytest

# Adding tests

The package already has a test folder with a test module

We can add for each function a test (there can be many tests for a function)



A screenshot of a code editor (VS Code) showing a Python test module named `test_pyadv_course.py`. The editor interface includes a sidebar with file navigation, a top bar with tabs and icons, and a bottom status bar.

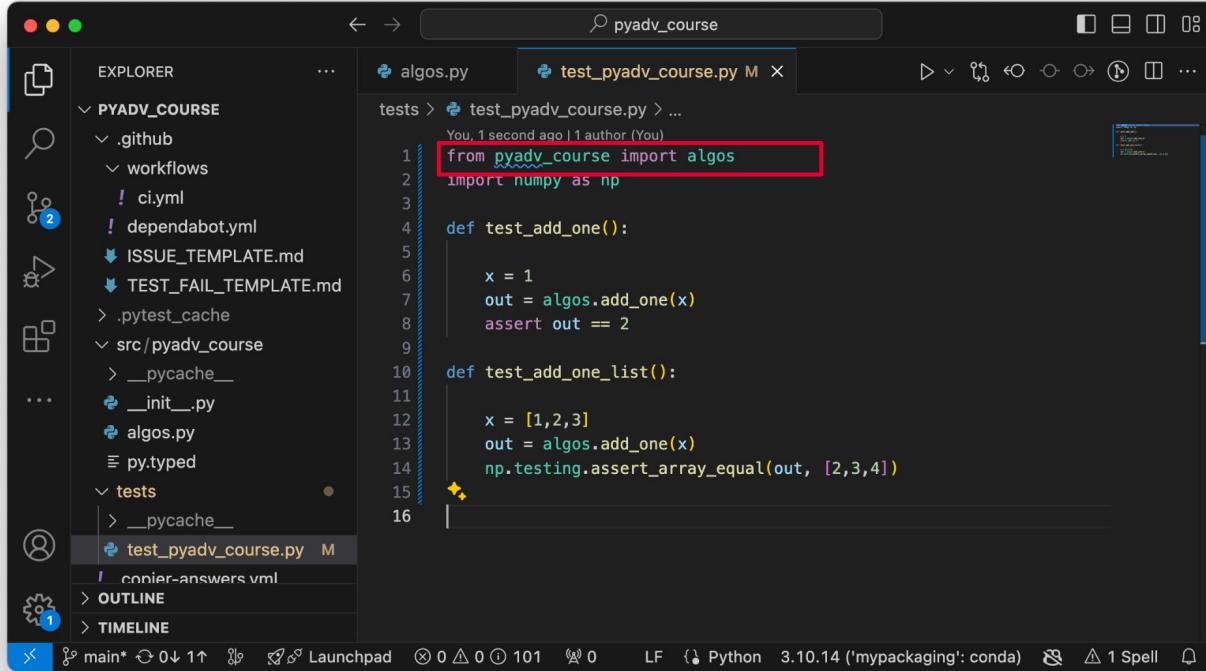
```
You, 1 second ago | 1 author (You)
from pyadv_course import algos
import numpy as np

def test_add_one():
    x = 1
    out = algos.add_one(x)
    assert out == 2

def test_add_one_list():
    x = [1,2,3]
    out = algos.add_one(x)
    np.testing.assert_array_equal(out, [2,3,4])
```

# Adding tests

Note: no relative import. **We test the installed package.**



The screenshot shows a dark-themed instance of VS Code. The left sidebar contains a file tree with a folder named 'PYADV\_COURSE' containing '.github', 'workflows', 'ci.yml', 'dependabot.yml', 'ISSUE\_TEMPLATE.md', 'TEST\_FAIL\_TEMPLATE.md', '.pytest\_cache', 'src/pyadv\_course' (which contains '\_\_pycache\_\_', '\_\_init\_\_.py', 'algos.py', and 'py.typed'), and 'tests' (which contains '\_\_pycache\_\_' and 'test\_pyadv\_course.py'). The right pane shows the code editor with a file named 'test\_pyadv\_course.py'. The code is as follows:

```
from pyadv_course import algos
import numpy as np

def test_add_one():
    x = 1
    out = algos.add_one(x)
    assert out == 2

def test_add_one_list():
    x = [1,2,3]
    out = algos.add_one(x)
    np.testing.assert_array_equal(out, [2,3,4])
```

A red box highlights the first line of code, 'from pyadv\_course import algos'. The status bar at the bottom of the screen shows the file path 'main\*', line numbers 0-11, and the Python version '3.10.14 ('mypackaging': conda)'.

# pytest

To automate our testing we use  
pytest

pytest is a Python package itself  
which

- can discover all tests files in a module
- run all tests
- report on success / failure
- automatically create series of tests e.g. for ranges of values

The screenshot shows a web browser displaying the pytest documentation at <https://docs.pytest.org/en/7.3.x/>. The page features the pytest logo and the tagline "pytest: helps you write better programs". It includes sections for "About pytest", "Contents", "About the project", and "A quick example". The "A quick example" section contains a code snippet for test\_sample.py:

```
# content of test_sample.py
def inc(x):
    return x + 1

def test_answer():
    assert inc(3) == 5
```

Below the code, instructions say "To execute it:" followed by a terminal command:

```
$ pytest
=====
platform linux -- Python 3.x.y, pytest-7.x.y, pluggy-1.x.y
rootdir: /home/sweet/project
collected 1 item
```

<https://docs.pytest.org/en/7.3.x/>

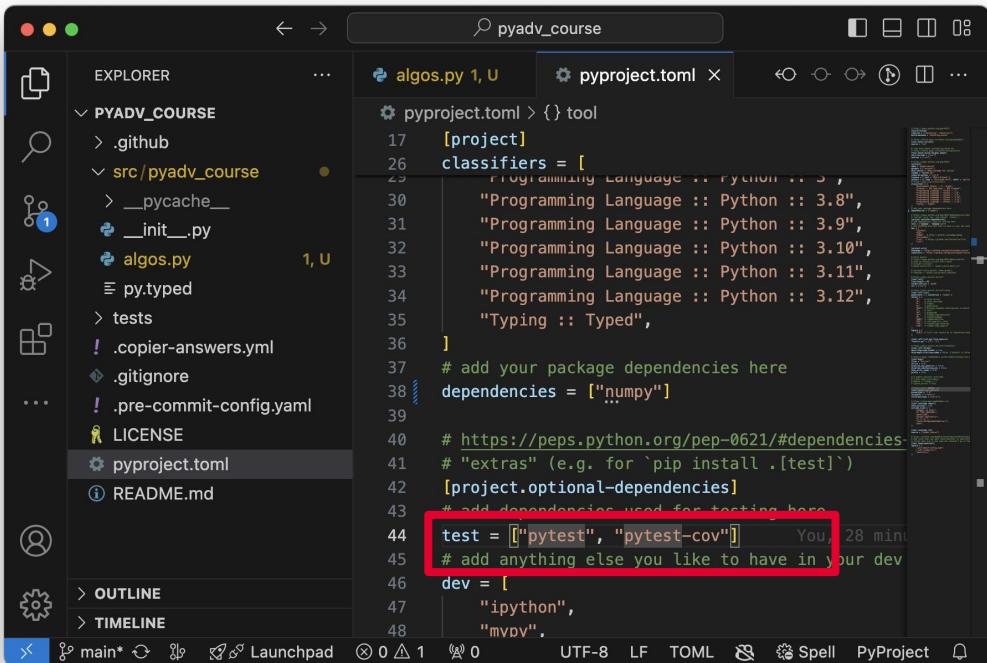
# Run pytest

First, install in env with:

**pip install pytest**

Or use your optional install!

**pip install ".[test]"**



The screenshot shows a code editor with a dark theme. On the left is the Explorer sidebar showing a project structure with files like .github, src/pyadv\_course, \_\_pycache\_\_, \_\_init\_\_.py, algos.py, py.typed, tests, .copier-answers.yml, .gitignore, .pre-commit-config.yaml, LICENSE, and README.md. The README.md file is currently selected. On the right is the main editor area displaying the contents of a pyproject.toml file. A red box highlights the 'test' section at the bottom of the file.

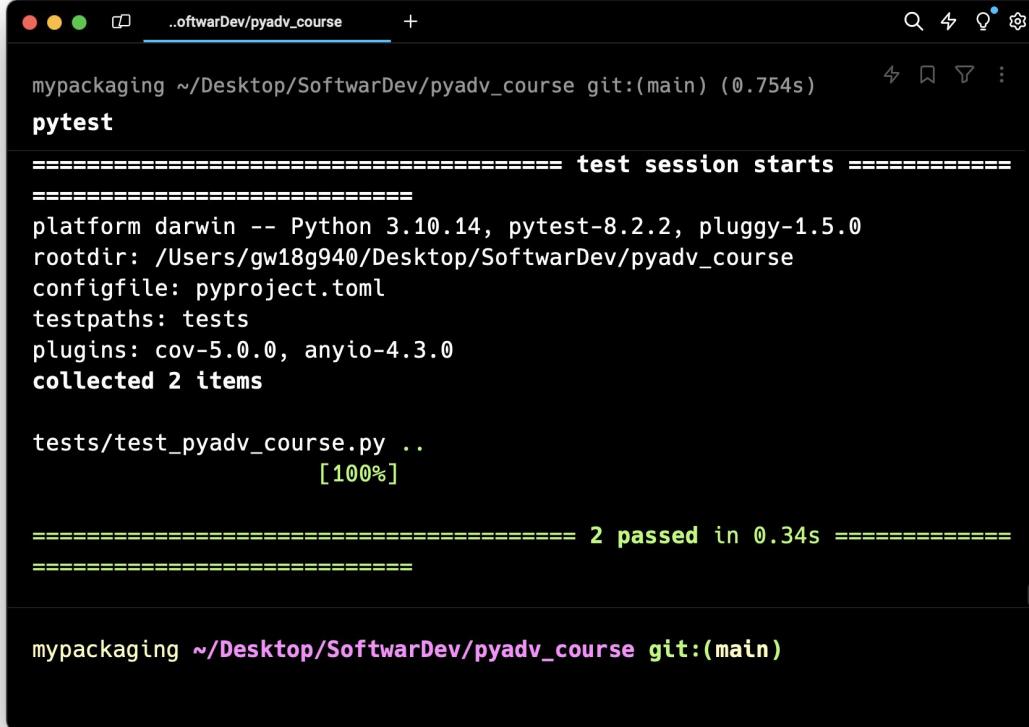
```
pyproject.toml
[project]
classifiers = [
    "Programming Language :: Python :: 3",
    "Programming Language :: Python :: 3.8",
    "Programming Language :: Python :: 3.9",
    "Programming Language :: Python :: 3.10",
    "Programming Language :: Python :: 3.11",
    "Programming Language :: Python :: 3.12",
    "Typing :: Typed",
]

# add your package dependencies here
dependencies = ["numpy"]

# 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",
]
```

# Run pytest

Then from inside project folder: **pytest**



```
myPackaging ~/Desktop/SoftwarDev/pyadv_course git:(main) (0.754s)
pytest
=====
platform darwin -- Python 3.10.14, pytest-8.2.2, pluggy-1.5.0
rootdir: /Users/gw18g940/Desktop/SoftwarDev/pyadv_course
configfile: pyproject.toml
testpaths: tests
plugins: cov-5.0.0, anyio-4.3.0
collected 2 items

tests/test_pyadv_course.py ..
    [100%]

=====
2 passed in 0.34s
=====

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

# Test parametrization

Here the test is parametrized with a decorator to run with:

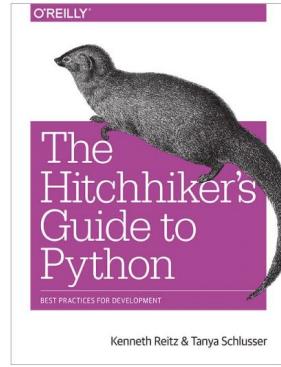
List to test: [1, 2] with expected results [2, 3] and

[1.1, 2.2] with expected results [2.1, 3.2]

# General rules

## O'Reilly Book

This guide is now available in tangible book form!



All proceeds are being directly donated to the [Django Girls](#) organization.

## Translations

[English](#)

[French](#)

[Chinese](#)

[Japanese](#)

[Korean](#)

[Filipino](#)

Testing your code is very important.

Getting used to writing testing code and running this code in parallel is now considered a good habit. Used wisely, this method helps to define your code's intent more precisely and have a more decoupled architecture.

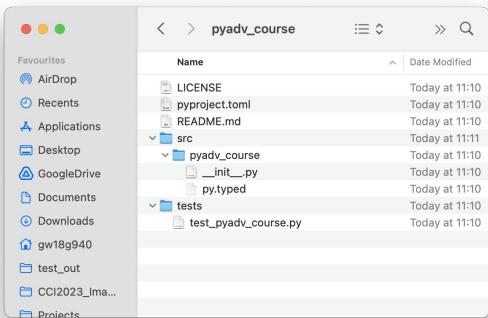
Some general rules of testing:

- A testing unit should focus on one tiny bit of functionality and prove it correct.
- Each test unit must be fully independent. Each test must be able to run alone, and also within the test suite, regardless of the order that they are called. The implication of this rule is that each test must be loaded with a fresh dataset and may have to do some cleanup afterwards. This is usually handled by `setUp()` and `tearDown()` methods.
- Try hard to make tests that run fast. If one single test needs more than a few milliseconds to run, development will be slowed down or the tests will not be run as often as is desirable. In some cases, tests can't be fast because they need a complex data structure to work on, and this data structure must be loaded every time the test runs. Keep these heavier tests in a separate test suite that is run by some scheduled task, and run all other tests as often as needed.
- Learn your tools and learn how to run a single test or a test case. Then, when developing a function inside a module, run this function's tests frequently, ideally automatically when you save the code.
- Always run the full test suite before a coding session, and run it again after. This will give you more confidence that you did not break anything in the rest of the code.
- It is a good idea to implement a hook that runs all tests before pushing code to a shared repository.

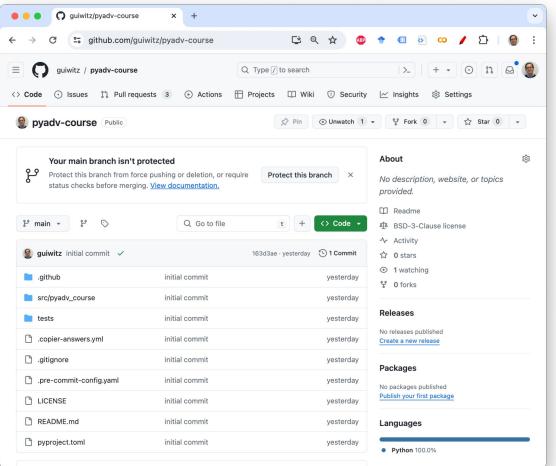
<https://docs.pythontutorial-guide.org/writing/tests/>

# Running test automatically: GitHub actions

# GitHub actions



push  
→



Run  
scripts  
upon a  
push  
→

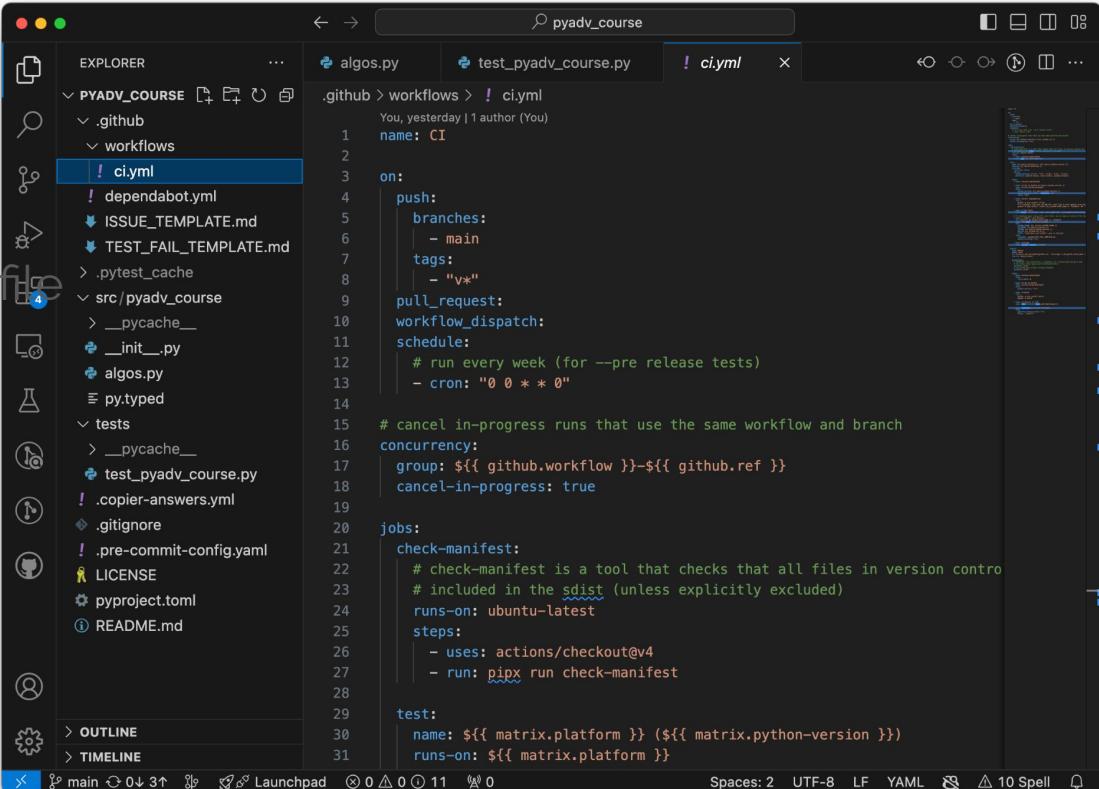


# Actions specified in yml file

File create by copier:

.github/workflows/ci.yml

The file is hidden on the regular system



A screenshot of a code editor showing the GitHub Actions configuration file `ci.yml`. The file is located in the `.github/workflows` directory of a project named `PYADV_COURSE`. The code editor interface includes an Explorer sidebar on the left showing other files like `algos.py`, `test_pyadv_course.py`, and `pyproject.toml`. The main pane displays the `ci.yml` content, which defines a workflow named `CI` triggered by pushes to the `main` branch and pull requests. It includes a weekly cron schedule and concurrency controls. The `jobs: check-manifest` section uses the `actions/checkout@v4` action and runs `pipx run check-manifest`. The `test:` section runs tests on a matrix of platforms and Python versions.

```
name: CI
on:
  push:
    branches:
      - main
    tags:
      - "v*"
  pull_request:
  workflow_dispatch:
  schedule:
    # run every week (for --pre release tests)
    - cron: "0 0 * * 0"
  # cancel in-progress runs that use the same workflow and branch
  concurrency:
    group: ${{ github.workflow }}-${{ github.ref }}
    cancel-in-progress: true
jobs:
  check-manifest:
    # check-manifest is a tool that checks that all files in version control
    # included in the sdist (unless explicitly excluded)
    steps:
      - uses: actions/checkout@v4
      - run: pipx run check-manifest
  test:
    name: ${matrix.platform} (${{ matrix.python-version }})
    runs-on: ${matrix.platform}
```

# Actions specified in yml file

Workflow name

When to run the workflow. Here it runs:

- after a push on main
- after a pull request
- via manual triggering
- at regular time intervals

Handle canceling (not necessary)

List of actual actions to take, aka **jobs**

Each subsection is a separate task for which one can specify the OS type and the actions to execute

```
name: CI

on:
  push:
    branches:
      - main
    tags:
      - "v*"
  pull_request:
  workflow_dispatch:
  schedule:
    # run every week (for --pre release tests)
    - cron: "0 0 * * 0"

# cancel in-progress runs that use the same workflow and branch
concurrency:
  group: ${{ github.workflow }}-${{ github.ref }}
  cancel-in-progress: true

jobs:
  check-manifest:
    # check-manifest is a tool that checks that all files in version control
    # included in the sdist (unless explicitly excluded)
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - run: pipx run check-manifest

  test:
    name: ${{ matrix.platform }} (${{ matrix.python-version }})
    runs-on: ${{ matrix.platform }}
```

# Specify job name and OS

Job running tests (naming is up to you)

Name of job (could be just a string) and OS to use e.g. "ubuntu-latest".

Here the name is formed using info from the matrix section

The tests will run on multiple OS (platform) and for multiple Python versions specified in the matrix section

```
20   jobs:
21     check-manifest:
22       # check-manifest is a tool that checks that all files in version control are
23       # included in the sdist (unless explicitly excluded)
24       runs-on: ubuntu-latest
25       steps:
26         - uses: actions/checkout@v4
27         - run: pipx run check-manifest
28
29   test:
30     name: ${{ matrix.platform }} (${{ matrix.python-version }})
31     runs-on: ${{{ matrix.platform }}}
32     strategy:
33       fail-fast: false
34       matrix:
35         python-version: ["3.8", "3.9", "3.10", "3.11", "3.12"]
36         platform: [ubuntu-latest, macos-latest, windows-latest]
37
38   steps:
39     - uses: actions/checkout@v4
40
41     - name: Set up Python ${{{ matrix.python-version }}}
42       uses: actions/setup-python@v4
43       with:
44         python-version: ${{{ matrix.python-version }}}
45         cache-dependency-path: "pyproject.toml"
46         cache: "pip"
47
48     - name: Install Dependencies
49       run: |
50         python -m pip install --U pip
51         # if running a cron job, we add the --pre flag to test against pre-releases
52         python -m pip install .[test] ${{{ github.event_name == 'schedule' && '--pre' || '' }}}
53
54     - name: Run Tests
55       run: pytest --color=yes --cov --cov-report=xml --cov-report=term-missing
56
57     # If something goes wrong with --pre tests, we can open an issue in the repo
58     - name: Report --pre Failures
59       if: failure() && github.event_name == 'schedule'
60       uses: JasonEtco/create-an-issue@v2
61       env:
62         GITHUB_TOKEN: ${{{ secrets.GITHUB_TOKEN }}}
63         PLATFORM: ${{{ matrix.platform }}}
64         PYTHON: ${{{ matrix.python-version }}}
65         RUN_ID: ${{{ github.run_id }}}
66         TITLE: "[test-bot] pip install --pre is failing"
67       with:
68         filename: .github/TEST_FAIL_TEMPLATE.md
69         update_existing: true
70
71     - name: Coverage
72       uses: codecov/codecov-action@v3
```

# Specify test job steps

This step checks out the repo (copies it). Entirely implemented as an action, a pre-made set of steps you can just use!

This step installs Python using an action + some options specified by with, for example the Python version (specified by matrix)

This step just runs some terminal commands directly. Here it updates pip and installs the package with the optional test dependencies

Finally it runs the tests (with some additional reporting options)

Additional steps create failure/coverage reports etc.

```
20 jobs:
21   check-manifest:
22     # check-manifest is a tool that checks that all files in version control are
23     # included in the sdist (unless explicitly excluded)
24     runs-on: ubuntu-latest
25     steps:
26       - uses: actions/checkout@v4
27       - run: pipx run check-manifest
28
29
30 test:
31   name: ${{ matrix.platform }} (${{ matrix.python-version }})
32   runs-on: ${{ matrix.platform }}
33   strategy:
34     fail-fast: false
35     matrix:
36       python-version: ["3.8", "3.9", "3.10", "3.11", "3.12"]
37       platform: [ubuntu-latest, macos-latest, windows-latest]
38
39   steps:
40     - uses: actions/checkout@v4
41
42     - name: Set up Python ${{ matrix.python-version }}
43       uses: actions/setup-python@v4
44       with:
45         python-version: ${{ matrix.python-version }}
46         cache-dependency-path: "pyproject.toml"
47         cache: "pip"
48
49     - name: Install Dependencies
50       run: |
51         python -m pip install --U pip
52         # if running a cron job, we add the --pre flag to test against pre-releases
53         python -m pip install .[test] ${{
54           github.event_name == 'schedule' && '--pre' || '' }}}
55
56     - name: Run Tests
57       run: pytest --color=yes --cov --cov-report=xml --cov-report=term-missing
58
59     # If something goes wrong with --pre tests, we can open an issue in the repo
60     - name: Report --pre Failures
61       if: failure() && github.event_name == 'schedule'
62       uses: JasonEtc0/create-an-issue@v2
63       env:
64         GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
65         PLATFORM: ${{ matrix.platform }}
66         PYTHON: ${{ matrix.python-version }}
67         RUN_ID: ${{
68           github.run_id
69           }}}
70         TITLE: "[test-bot] pip install --pre is failing"
71       with:
72         filename: .github/TEST_FAIL_TEMPLATE.md
73         update_existing: true
74
75     - name: Coverage
76       uses: codecov/codecov-action@v3
```

# Actions specified in yml file

In the Actions tab of GitHub you can see the status of actions:

success, running, failed

You can select a run to see more details

The screenshot shows the GitHub Actions tab for the repository 'guiwitz/pyadv-course'. The left sidebar lists 'Actions' (selected), 'Code', 'Issues', 'Pull requests', 'Actions' (link to this page), 'Projects', 'Wiki', 'Security', 'Insights', and 'Settings'. Under 'Actions', there are sections for 'All workflows' (selected), 'CI' (Dependabot Updates), 'Management' (Caches, Attestations, Runners), and 'Dependabot' (Updates). The main area displays 'All workflows' and '5 workflow runs'. Each run is listed with its name, event, status, branch, actor, and timestamp.

Workflow Run	Event	Status	Branch	Actor	Timestamp
ci(dependabot): bump softprops/action-gh-release from 1 to 2	CI #4: Pull request #3 opened by dependabot (bot)	dependabot/github_actions..	main	dependabot/github_actions..	2 days ago 2m 15s
ci(dependabot): bump codecov/codecov-action from 3 to 4	CI #3: Pull request #2 opened by dependabot (bot)	dependabot/github_actions..	main	dependabot/github_actions..	2 days ago 1m 45s
ci(dependabot): bump actions/setup-python from 4 to 5	CI #2: Pull request #1 opened by dependabot (bot)	dependabot/github_actions..	main	dependabot/github_actions..	2 days ago 1m 2s
github_actions in / - Update #837762849	Dependabot Updates #1: by dependabot (bot)	main	main	dependabot/github_actions..	2 days ago 42s
initial commit	CI #1: Commit 163d3ae pushed by guiwitz	main	main	dependabot/github_actions..	2 days ago 1m 6s

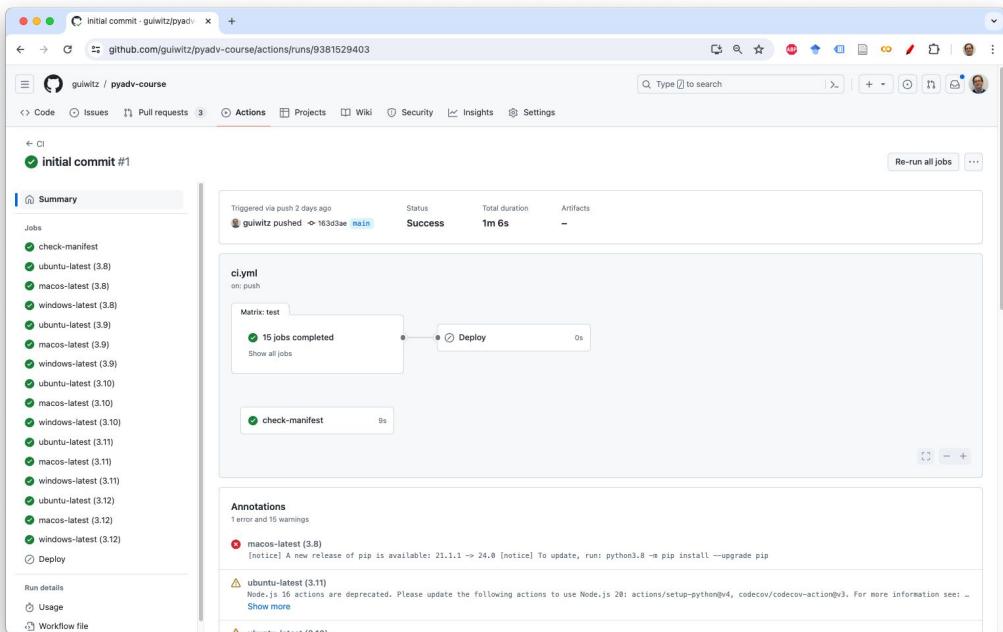
# Actions specified in yml file

We see the combination of OS and Python that were tested

The test jobs were run

A second part called Deploy was not executed

<https://github.com/guiwitz/pyadv-course/actions/runs/5215870500/jobs/9413950714>



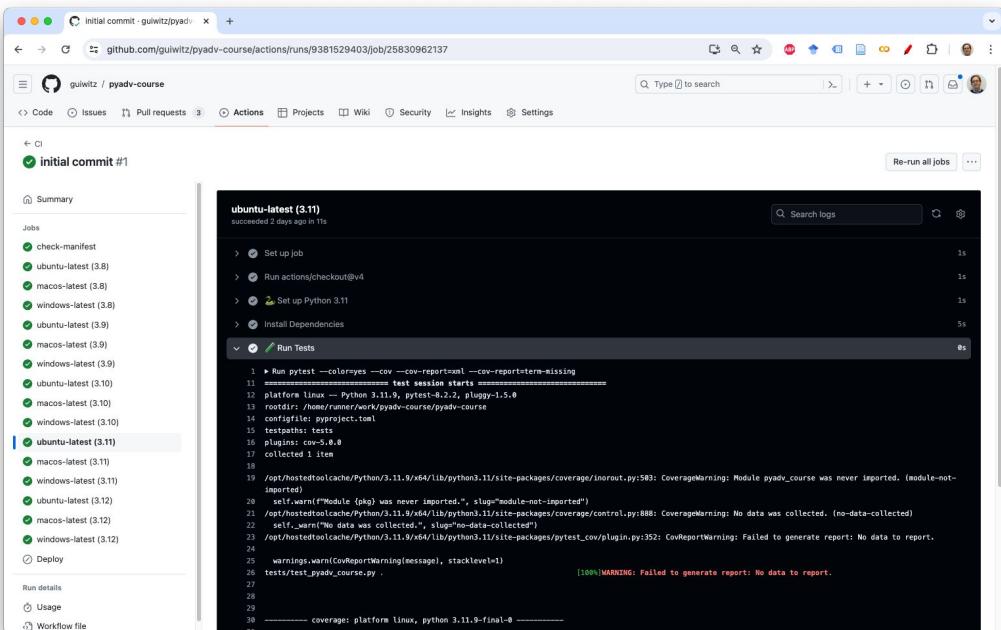
# Actions specified in yml file

We see the combination of OS and Python that were tested

The test jobs were run

A second part called Deploy was not executed

Terminal details can be checked for each run



The screenshot shows a GitHub Actions run page for a repository named 'pyadv-course'. The run is titled 'initial commit #1' and was completed 2 days ago. The 'ubuntu-latest (3.11)' job is highlighted, showing its status as 'succeeded'. The job log displays the command 'Run pytest --color=yes --cov --cov-report=xml --cov-report=term-missing' and its output, which includes several 'CoverageWarning' messages indicating that the module 'pyadv\_course' was never imported. The log ends with a 'WARNING: Failed to generate report: No data to report.' message.

```
ubuntu-latest (3.11)
succeeded 2 days ago in 11s

> ⚡ Set up job
> ⚡ Run actions/checkout@v4
> ⚡ Set up Python 3.11
> ⚡ Install Dependencies
> ⚡ Run Tests
1 ▶ Run pytest --color=yes --cov --cov-report=xml --cov-report=term-missing
11 11
12 12
13 platform: linux - Python 3.11.9, pytest-6.2.2, py-3.10.0
13 rootdir: /home/rnrunner/work/pyadv-course/pyadv-course
14 configfile: pyproject.toml
15 testpaths: tests
16 plugins: cov-5.0.0
17 collected 1 item
18
19 /opt/hostedtoolcache/Python/3.11.9/x64/lib/python3.11/site-packages/coverage/inorout.py:503: CoverageWarning: Module pyadv_course was never imported. (module-not-imported)
20 self.warn(f"Module {pkg} was never imported.", slug="module-not-imported")
21 /opt/hostedtoolcache/Python/3.11.9/x64/lib/python3.11/site-packages/coverage/control.py:886: CoverageWarning: No data was collected. (no-data-collected)
22 self._warn("No data was collected.", slug="no-data-collected")
23 /opt/hostedtoolcache/Python/3.11.9/x64/lib/python3.11/site-packages/pytest_cov/plugin.py:352: CovReportWarning: Failed to generate report: No data to report.
24
25 warnings.warn(CovReportWarning(message), stacklevel=1)
26 tests/test_pyadv_course.py
27
28
29 [100%]WARNING: Failed to generate report: No data to report.
30
31 ----- coverage: platform linux, python 3.11.9-final-0 -----
```

Side note: Another type of test, typing

# Python type inference

Unlike many other languages where one states:

```
float a = 3.5
```

in Python variable types are inferred

Good: simple to develop and write

Bad: sometimes silent errors creep in

```
[1]: a = 3.5  
      type(a)
```

```
[1]: float
```

```
[2]: b = 5  
      type(b)
```

```
[2]: int
```

# How should a function be used?

Probably with numbers ?

```
def multiply_fun(a, b):  
    out = a * b  
    return out
```

# How should a function be used?

Probably with numbers ?

```
: multiply_fun(3, 4)
```

```
: 12
```

# How should a function be used?

Probably with numbers ? No error with a letter, but might create a problem later...

```
: multiply_fun(3, 4)
```

```
: 12
```

```
: multiply_fun(3, 't')
```

```
: 'ttt'
```

# Python typing

Since a few version of Python, one can now “type” functions, i.e. specifying what inputs and outputs should be. Using the following syntax:

```
def multiply_fun(a: float, b: float) -> float:  
    out = a * b  
  
    return out
```

# Python typing

After annotation, the function still doesn't fail. However tools allow to analyze the function and its usage and warn us of inconsistencies.

```
def multiply_fun(a: float, b: float) -> float:  
    out = a * b  
    return out
```

```
multiply_fun(3, 't')
```

```
'ttt'
```

# mypy

Unlike for test, no need to write additional function

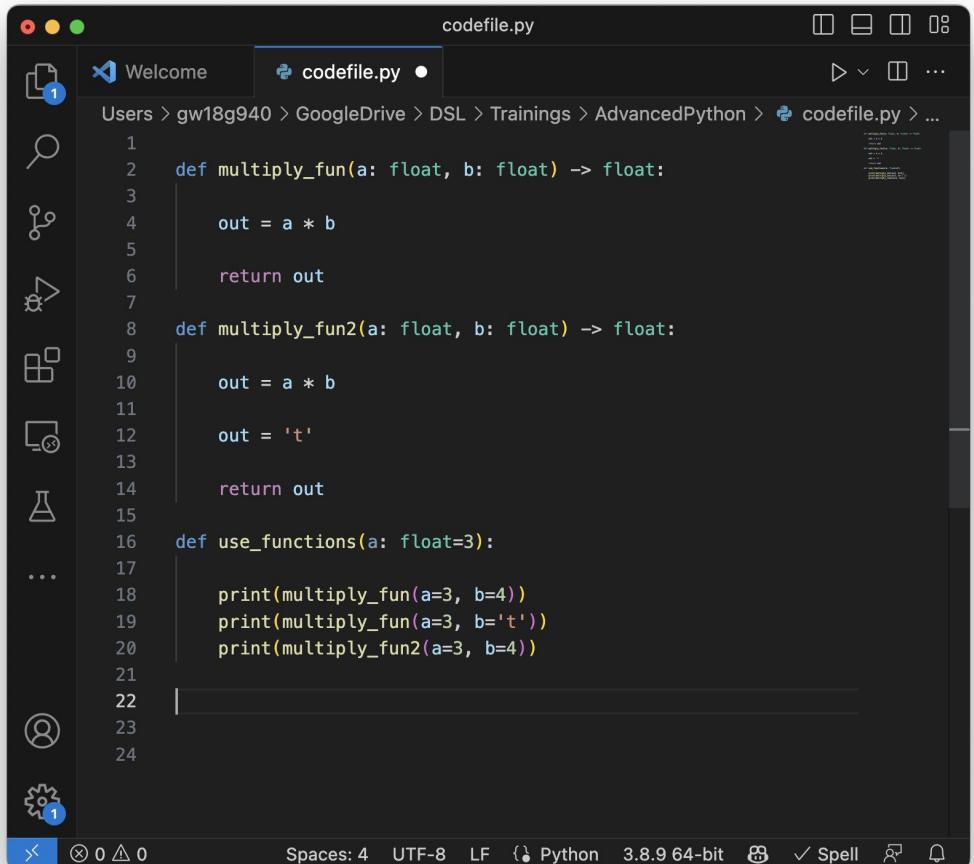
Tools like mypy go through code and check for type inconsistencies

Here we have:

- bad inputs in `use_functions`
- bad output in `multiply_fun2`

Install mypi to find those issues:

`pip install mypy`



The screenshot shows a code editor window titled "codefile.py". The file contains the following Python code:

```
1  def multiply_fun(a: float, b: float) -> float:
2      out = a * b
3      return out
4
5  def multiply_fun2(a: float, b: float) -> float:
6      out = a * b
7      out = 't'
8      return out
9
10 def use_functions(a: float=3):
11     print(multiply_fun(a=3, b=4))
12     print(multiply_fun(a=3, b='t'))
13     print(multiply_fun2(a=3, b=4))
14
15
16
17
18
19
20
21
22
23
24
```

The code defines three functions: `multiply_fun`, `multiply_fun2`, and `use_functions`. The `multiply_fun` and `use_functions` functions are annotated with type hints. The `multiply_fun2` function has a bug where it returns a string instead of a float. The `use_functions` function calls these functions and prints their results.

# mypy

The screenshot displays a macOS desktop environment with a code editor and a terminal window.

**Code Editor:** The code editor window is titled "codefile.py". It contains the following Python code:

```
1  def multiply_fun(a: float, b: float) -> float:
2
3      out = a * b
4
5      return out
6
7
8  def multiply_fun2(a: float, b: float) -> float:
9
10     out = a * b
11
12     out = 't'
13
14     return out
15
16 def use_functions(a: float=3):
17
18     print(multiply_fun(a=3, b=4))
19     print(multiply_fun(a=3, b='t'))
20     print(multiply_fun2(a=3, b=4))
21
22
23
24
```

**Terminal:** The terminal window shows the following output:

```
(advpython) ~/GoogleDrive/DSL/Trainings/AdvancedPython (0.736s)
pip install mypy
Requirement already satisfied: typing-extensions>=3.10 in /Users/gw18g940/mambaforge/envs/advpython/lib/python3.10/site-packages (from mypy) (4.6.3)
Requirement already satisfied: mypy-extensions>=1.0.0 in /Users/gw18g940/mambaforge/envs/advpython/lib/python3.10/site-packages (from mypy) (1.0.0)
Requirement already satisfied: tomli>=1.1.0 in /Users/gw18g940/mambaforge/envs/advpython/lib/python3.10/site-packages (from mypy) (2.0.1)

(advpython) ~/GoogleDrive/DSL/Trainings/AdvancedPython (0.252s)
mypy codefile.py
codefile.py:11: error: Incompatible types in assignment (expression has type "str", variable has type "float") [assignment]
codefile.py:18: error: Argument "b" to "multiply_fun" has incompatible type "str"; expected "float" [arg-type]
Found 2 errors in 1 file (checked 1 source file)

(advpython) ~/GoogleDrive/DSL/Trainings/AdvancedPython
```

The terminal output indicates that the code contains two errors related to incompatible type assignments and arguments.

Back to packaging

# Releasing a package

After packaging we want to distribute our software via PyPi

One can build everything locally and upload, but:

- all the work is manual
- no automatic check on tests
- no way to build for other platforms

So we build directly on GitHub and distribute from there.

Can be done using Actions

# Updating our workflow

In the GitHub actions workflow, we also check if the commit is tagged

```
name: CI  
  
on:  
  push:  
    branches:  
      - main  
    tags:  
      - "v*"  
  pull_request:
```

# Deploy job

- Only works:
  - if tests are run
- if previous jobs are successful
  - if the commit was tagged
  - if it's not a scheduled run

Only works:

```
.github > workflows > ! ci.yml
20   jobs:
21     deploy:
22       name: Deploy
23       needs: test
24       if: success() && startsWith(github.ref, 'refs/tags/') && github.event_name != 'schedule'
25       runs-on: ubuntu-latest
26
27       permissions:
28         # IMPORTANT: this permission is mandatory for trusted publishing on PyPi
29         # see https://docs.pypi.org/trusted-publishers/
30         id-token: write
31         # This permission allows writing releases
32         contents: write
33
34       steps:
35         - uses: actions/checkout@v4
36           with:
37             fetch-depth: 0
38
39         - name: 🐍 Set up Python
40           uses: actions/setup-python@v4
41           with:
42             python-version: "3.x"
43
44         - name: 🧑 Build
45           run:
46             - python -m pip install build
47             - python -m build
48
49         - name: 📦 Publish to PyPI
50           uses: pypa/gh-action-pypi-publish@release/v1
51
52         - uses: softprops/action-gh-release@v1
53           with:
54             generate_release_notes: true
55             files: './dist/*'
```

# Deploy job

Give GitHub writing rights  
on your repo to be able to  
make a release

```
.github > workflows > ! ci.yml
20   jobs:
21     deploy:
22       name: Deploy
23       needs: test
24       if: success() && startsWith(github.ref, 'refs/tags/') && github.event_name != 'schedule'
25       runs-on: ubuntu-latest
26
27     permissions:
28       # IMPORTANT: this permission is mandatory for trusted publishing on PyPi
29       # see https://docs.pypi.org/trusted-publishers/
30       id-token: write
31       # This permission allows writing releases
32       contents: write
33
34     steps:
35       - uses: actions/checkout@v4
36         with:
37           fetch-depth: 0
38
39       - name: 🐍 Set up Python
40         uses: actions/setup-python@v4
41         with:
42           python-version: "3.x"
43
44       - name: 🧑 Build
45         run:
46           python -m pip install build
47           python -m build
48
49       - name: 📦 Publish to PyPI
50         uses: pypa/gh-action-pypi-publish@release/v1
51
52       - uses: softprops/action-gh-release@v1
53         with:
54           generate_release_notes: true
55           files: './dist/*'
```

# Deploy job

- Check out the repo
- Setup Python
- “Build” the package i.e. wrap it in a conform way
- Upload the package to PyPi

```
.github > workflows > ! ci.yml
20   jobs:
21     deploy:
22       name: Deploy
23       needs: test
24       if: success() && startsWith(github.ref, 'refs/tags/') && github.event_name != 'schedule'
25       runs-on: ubuntu-latest
26
27     permissions:
28       # IMPORTANT: this permission is mandatory for trusted publishing on PyPi
29       # see https://docs.pypi.org/trusted-publishers/
30       id-token: write
31       # This permission allows writing releases
32       contents: write
33
34     steps:
35       - uses: actions/checkout@v4
36         with:
37           fetch-depth: 0
38
39       - name: 🐍 Set up Python
40         uses: actions/setup-python@v4
41         with:
42           python-version: "3.x"
43
44       - name: 🤖 Build
45         run:
46           - python -m pip install build
47           - python -m build
48
49       - name: 📦 Publish to PyPI
50         uses: pypa/gh-action-pypi-publish@release/v1
51
52       - uses: softprops/action-gh-release@v1
53         with:
54           generate_release_notes: true
55           files: './dist/*'
```

# TestPyPi

To avoid polluting PyPi with our test packages, we use here TestPyPi which is a PyPi sandbox.

Please add the following in the Publish section:

**with:**

**repository-url: <https://test.pypi.org/legacy/>**

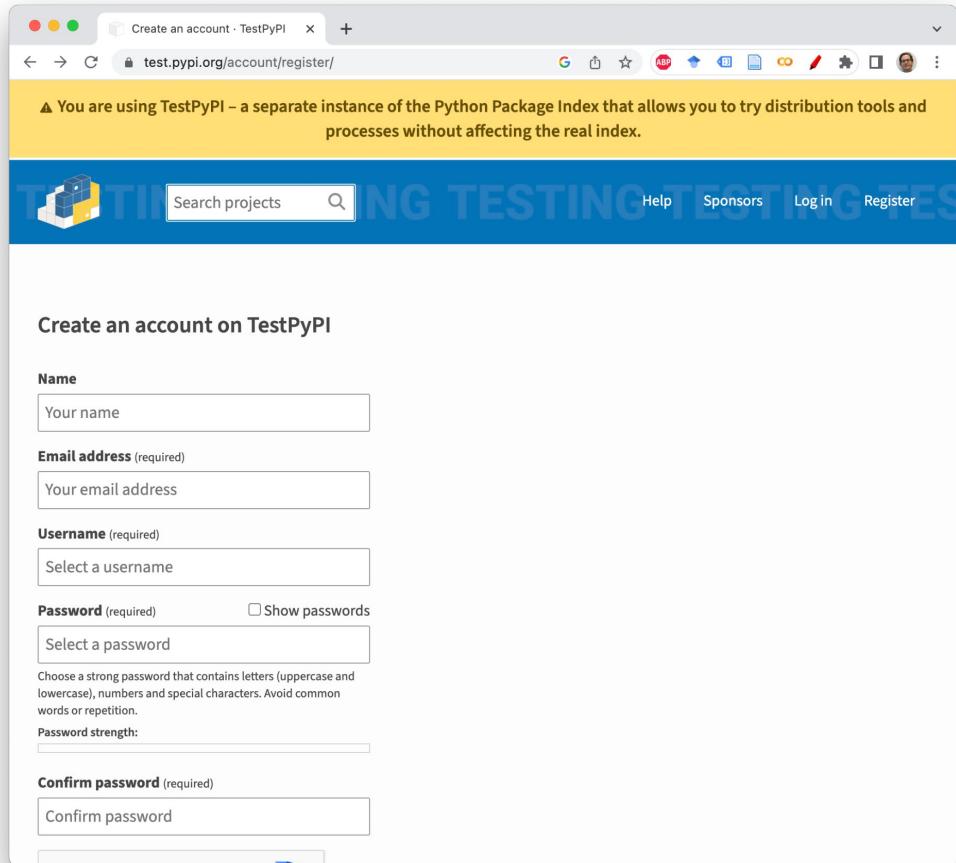
```
- name: Publish to TestPyPI
  uses: pypa/gh-action-pypi-publish@release/v1
  with:
    repository-url: https://test.pypi.org/legacy/
```

# Create account on TestPyPi

This is a sandbox, so the account doesn't matter

However create something you remember as it's useful to use for tests before release

<https://test.pypi.org/account/register/>



The screenshot shows a web browser window for 'Create an account - TestPyPI'. The URL in the address bar is 'test.pypi.org/account/register/'. A yellow banner at the top states: '⚠ You are using TestPyPI – a separate instance of the Python Package Index that allows you to try distribution tools and processes without affecting the real index.' Below the banner, the TestPyPi logo is visible, along with a search bar containing 'Search projects' and a magnifying glass icon. On the right side of the header, there are links for 'Help', 'Sponsors', 'Log In', and 'Register'. The main content area is titled 'Create an account on TestPyPI'. It contains several input fields: 'Name' (placeholder 'Your name'), 'Email address (required)' (placeholder 'Your email address'), 'Username (required)' (placeholder 'Select a username'), 'Password (required)' (placeholder 'Select a password'), and 'Confirm password (required)' (placeholder 'Confirm password'). There is also a checkbox labeled 'Show passwords' and a note about password strength. The background features a blue banner with the words 'TESTING TESTING TESTING' repeated.

# Create account on TestPyPi

Login

Go to account settings

The screenshot shows a web browser window titled "Your projects - TestPyPi" at the URL [test.pypi.org/manage/projects/](https://test.pypi.org/manage/projects/). The page has a yellow header bar with the text: "⚠ You are using TestPyPi – a separate instance of the Python Package Index that allows you to try distribution tools and processes without affecting the real index." Below this, a message encourages enabling two-factor authentication. The main content area features a search bar and a sidebar with links for "Your account" (including "Your projects", "Your organizations", "Account settings", and "Publishing"). On the right, there's a sidebar with links for "Your projects", "Your organizations", "Account settings" (which is highlighted with a red box), "Public profile", "Help", "Sponsors", and "Log out". Two project cards are listed under "Your projects": "microfilm" (SOLE OWNER, last released Jul 12, 2022) and "morphodynamics" (SOLE OWNER, last released Jan 31, 2022). At the bottom, there are links for "Help", "About PyPI", and "Contributing to PyPI".

# Create account on TestPyPi

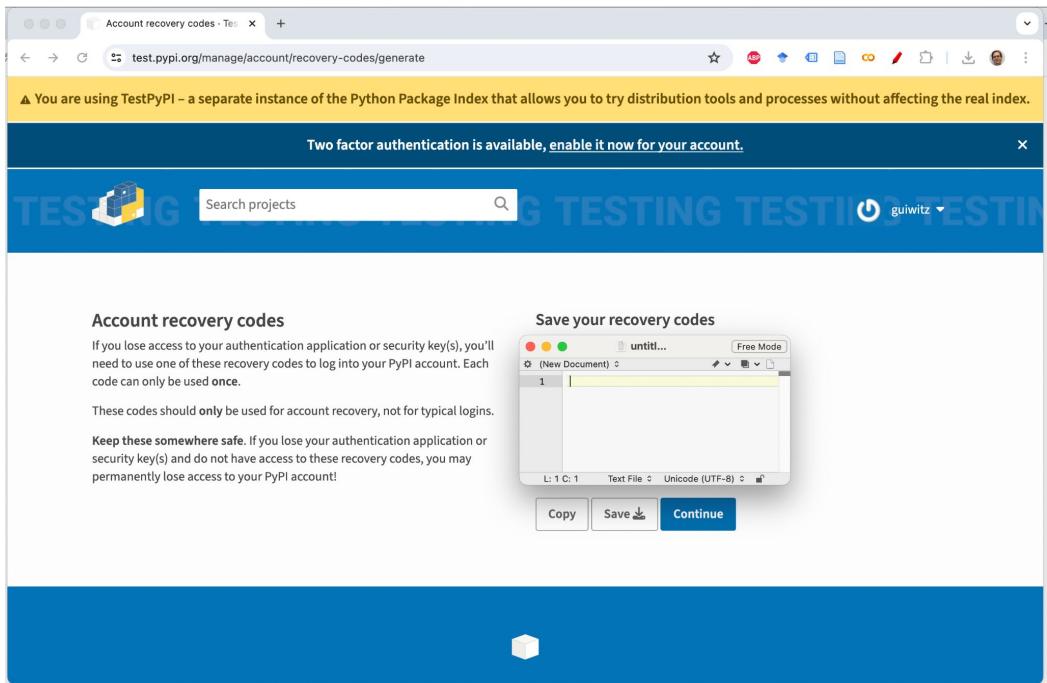
Login

Go to account settings

Enable two-factor authentication:

1. Save recovery codes

2. Use e.g. Authenticator app with  
QRCode (“Add 2FA with  
authentication application”)



# Create account on TestPyPi

Login

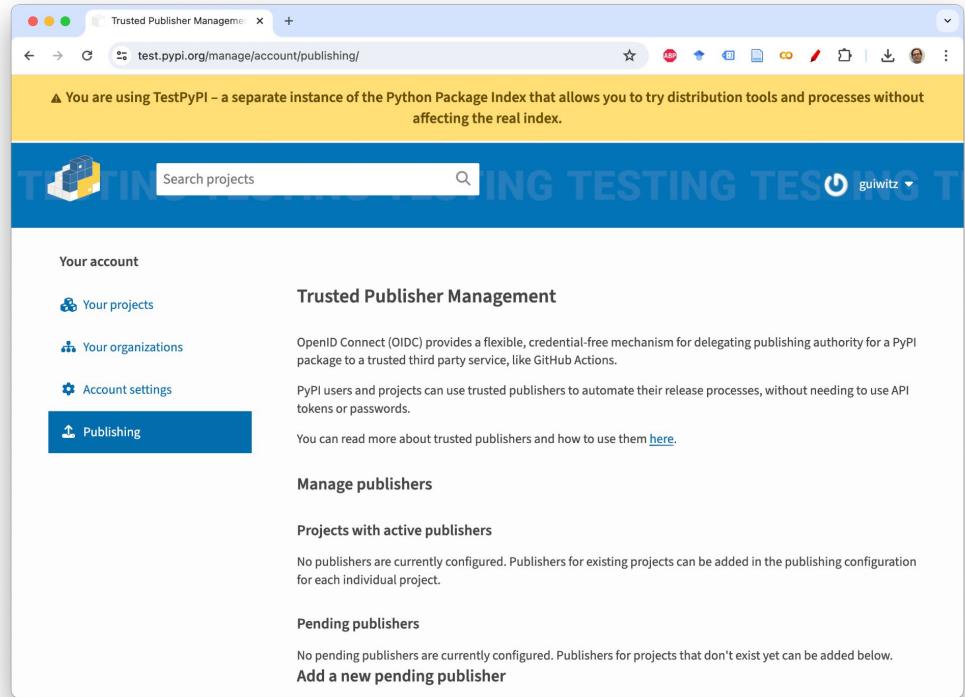
Go to account settings

Enable two-factor authentication:

1. Save recovery codes

2. Use e.g. Authenticator app with  
QRCode (“Add 2FA with  
authentication application”)

Go to Publishing tab



# Create account on TestPyPi

Login

Go to account settings

Enable two-factor authentication:

1. Save recovery codes

2. Use e.g. Authenticator app with  
QRCode (“Add 2FA with  
authentication application”)

Go to Publishing tab

Fill the form

⚠ You are using TestPyPi – a separate instance of the Python Package Index that allows you to try distribution tools and processes without affecting the real index.

**GitHub**   **GitLab**   **Google**   **ActiveState**

Read more about GitHub Actions' OpenID Connect support [here](#).

**PyPI Project Name** (required)  
pyadv-course

The project (on PyPI) that will be created when this publisher is used

**Owner** (required)  
guiwitz

The GitHub organization name or GitHub username that owns the repository

**Repository name** (required)  
pyadv-course

The name of the GitHub repository that contains the publishing workflow

**Workflow name** (required)  
ci.yml

The filename of the publishing workflow. This file should exist in the `.github/workflows/` directory in the repository configured above.

**Environment name** (optional)  
release

The name of the [GitHub Actions environment](#) that the above workflow uses for publishing. This should be configured under the repository's settings. While not required, a dedicated publishing environment is strongly encouraged, especially if your repository has maintainers with commit access who shouldn't have PyPI publishing access.

**Add**

# Finally we can try to release

We can now add a version tag to the repo and push it to GitHub. With that version tag, the publication to TestPyPi will happen:

Add a new release tag starting with v:

```
git tag -a v0.1.2 -m "v0.1.2"
```

Push everything from VSCode

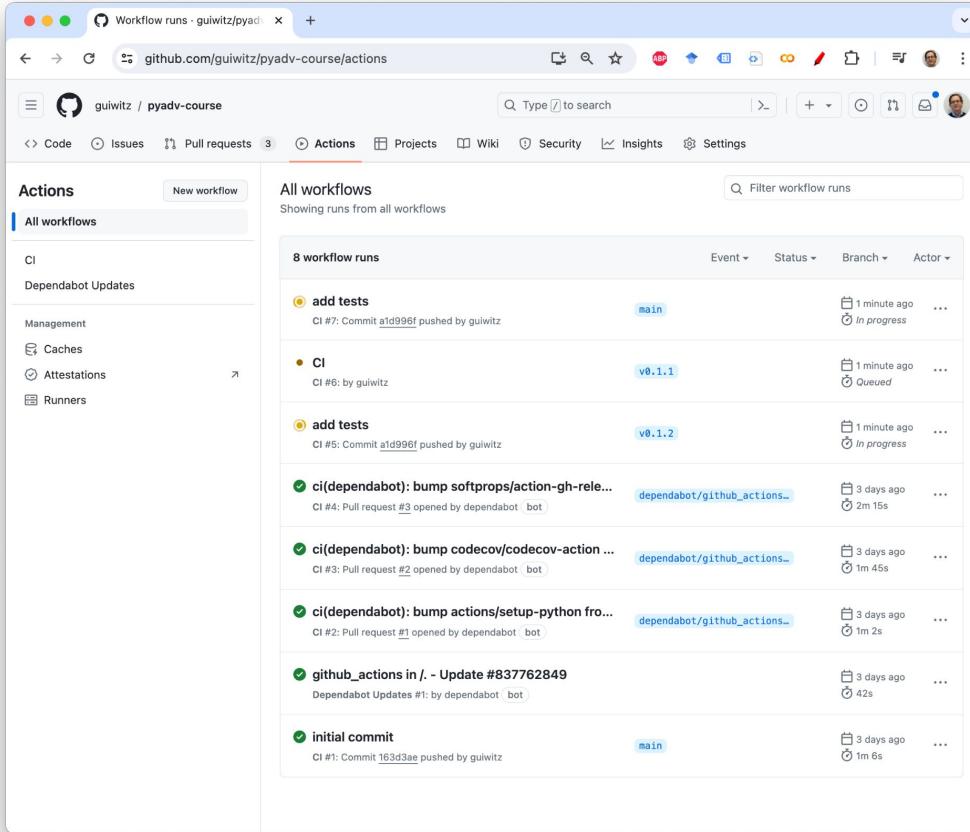
Push your tag:

```
git push --follow-tags
```



```
.github > workflows > ! ci.yml
20   jobs:
74     deploy:
75       name: Deploy
76       needs: test
77       if: success() && startsWith(github.ref, 'refs/tags/') && github.event_name != 'schedule'
78       runs-on: ubuntu-latest
79
80     permissions:
81       # IMPORTANT: this permission is mandatory for trusted publishing on PyPi
82       # see https://docs.pypi.org/trusted-publishers/
83       id-token: write
84       # This permission allows writing releases
85       contents: write
86
87     steps:
88       - uses: actions/checkout@v4
89         with:
90           fetch-depth: 0
91
92       - name: Set up Python
93         uses: actions/setup-python@v4
94         with:
95           python-version: "3.x"
96
97       - name: Build
98         run:
99           | python -m pip install build
100          | python -m build
101
102       - name: Publish to PyPI
103         uses: pypa/gh-action-pypi-publish@release/v1
104
105       - uses: softprops/action-gh-release@v1
106         with:
107           generate_release_notes: true
108           files: './dist/*'
```

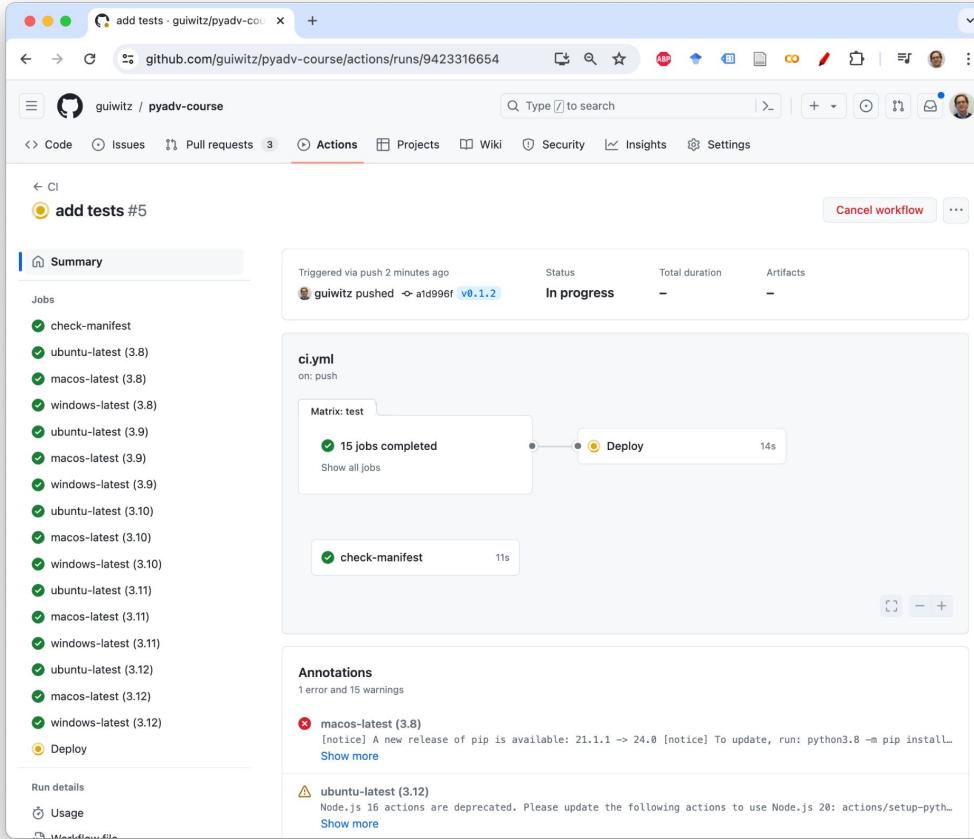
# Finally we can try to release



The screenshot shows the GitHub Actions workflow runs page for the repository `guiwitz/pyadv-course`. The page displays 8 workflow runs across various events, statuses, branches, and actors.

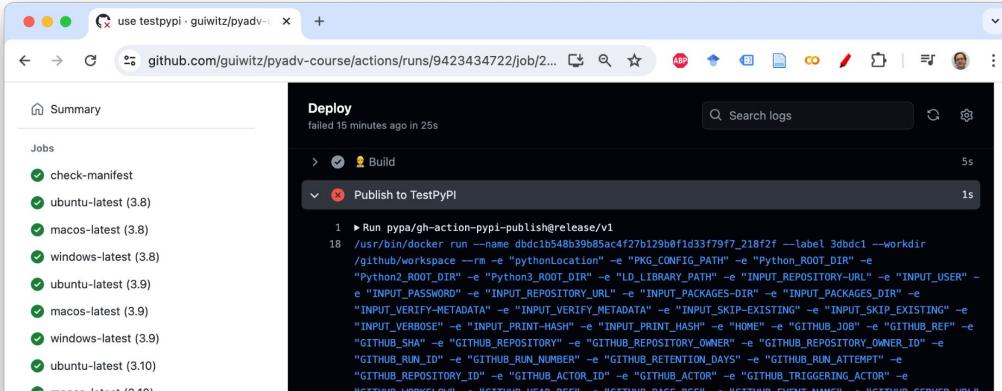
Event	Status	Branch	Actor
CI #7: Commit <a href="#">a1d996f</a> pushed by guiwitz	In progress	main	...
CI #6: by guiwitz	Queued	v0.1.1	...
CI #5: Commit <a href="#">a1d996f</a> pushed by guiwitz	In progress	v0.1.2	...
CI #4: Pull request #3 opened by dependabot · bot	Completed	dependabot/github_actions_...	3 days ago 2m 15s
CI #3: Pull request #2 opened by dependabot · bot	Completed	dependabot/github_actions_...	3 days ago 1m 45s
CI #2: Pull request #1 opened by dependabot · bot	Completed	dependabot/github_actions_...	3 days ago 1m 2s
Dependabot Updates #1: by dependabot · bot	Completed	...	3 days ago 42s
CI #1: Commit <a href="#">163d3ae</a> pushed by guiwitz	Completed	main	...

# Finally we can try to release



# Finally we can try to release... and fail

Name of package is already taken or close to a taken one!



```

20 Token request failed: the server refused the request for the following reasons:
21
22 * `invalid-payload`: The name 'pyadv-course' is too similar to an existing project. See
https://test.pypi.org/help/#project-name for more information.
23
24 This generally indicates a trusted publisher configuration error, but could
25 also indicate an internal error on GitHub or PyPI's part.
26
27
28 The claims rendered below are **for debugging purposes only**. You should **not***
29 use them to configure a trusted publisher unless they already match your expectations.
30
31 If a claim is not present in the claim set, then it is rendered as 'MISSING'.

```

# Finally we can try to release... and fail

A screenshot of a web browser showing the TestPyPI search results for the package 'pyadvcourse'. The URL is `test.pypi.org/search/?q=pyadvcourse`. The page header says, "You are using TestPyPI – a separate instance of the Python Package Index that allows you to try distribution tools and processes without affecting the real index." The search bar shows 'pyadvcourse'. The results section displays one project: 'pyadvcourse 0.2.1' with the description 'Demo package for course'. The package was released on Jun 9, 2023. On the left, there is a sidebar titled 'Filter by classifier' with various checkboxes checked: Framework, Topic, Development Status, License, Programming Language, Operating System, Environment, Intended Audience, Natural Language, and Typing.

Search results - TestPyPI

test.pypi.org/search/?q=pyadvcourse

You are using TestPyPI – a separate instance of the Python Package Index that allows you to try distribution tools and processes without affecting the real index.

pyadvcourse

TESTING TESTING TESTING TEST

Filter by classifier

Framework

Topic

Development Status

License

Programming Language

Operating System

Environment

Intended Audience

Natural Language

Typing

1 project for "pyadvcourse"

pyadvcourse 0.2.1

Demo package for course

Jun 9, 2023

Order by Relevance

Help About PyPI Contributing to PyPI Using PyPI

Installing packages PyPI Blog Bugs and feedback Code of conduct

# Houraaaaa

A screenshot of a GitHub Actions CI run details page. The URL is [use testpypi · guiwitz/pyadv-course · Actions](https://github.com/guiwitz/pyadv-course/actions/runs/9423618069). The run summary shows a single job named "use testpypi #11" triggered via push 2 minutes ago by "guiwitz pushed". The status is Success, total duration is 2m 2s, and there are no artifacts. The job matrix for "ci.yml" on "on: push" shows 15 jobs completed and one pending ("Deploy"). Below the matrix, a "check-manifest" step is shown with a duration of 8s. The annotations section shows 1 error and 16 warnings. A warning for "ubuntu-latest (3.9)" mentions deprecated Node.js actions. The sidebar includes sections for "Summary", "Jobs", "Annotations", "Run details", and "Usage".

A screenshot of the TestPyPI project page for "pyadv-course 0.1.3". The URL is [pyadv-course · TestPyPI](https://test.pypi.org/project/pyadv-course/). The page header indicates "TESTING TESTING TESTING TEST". The main content area shows the package name "pyadv-course 0.1.3" and the command "pip install -i https://test.pypi.org/simple/ pyadv-course". It states "Released: less than a minute ago". Below this, a "Demo package for course." is described. The "Project description" tab is selected, showing the project name "pyadv-course", release history, download files, and verified details (verified by PyPI). The "Maintainers" section lists "guiwitz". The "Unverified details" section is present at the bottom.