Python packaging, Part 2

Guillaume Witz
Data Science Lab, University of Bern

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

DSL

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

DSL

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

```python
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
```

```
3
```

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
```

```
2
```

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

AssertionError:
```

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

      assert a in mylist
```

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

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

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

AssertionError:
```

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]:
```python
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                                Traceback (most
recent call last)
Cell In[61], line 1
----> 1 assert out == 3, 'Thresholding not working'

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

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/uti
ls.py:862, in assert_array_compare(comparison, x, y, err_msg, verbose, header, precisio
n, 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/uti
ls.py:862, in assert_array_compare(comparison, x, y, err_msg, verbose, header, precisio
n, 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

DSL

```
[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

```python
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
```

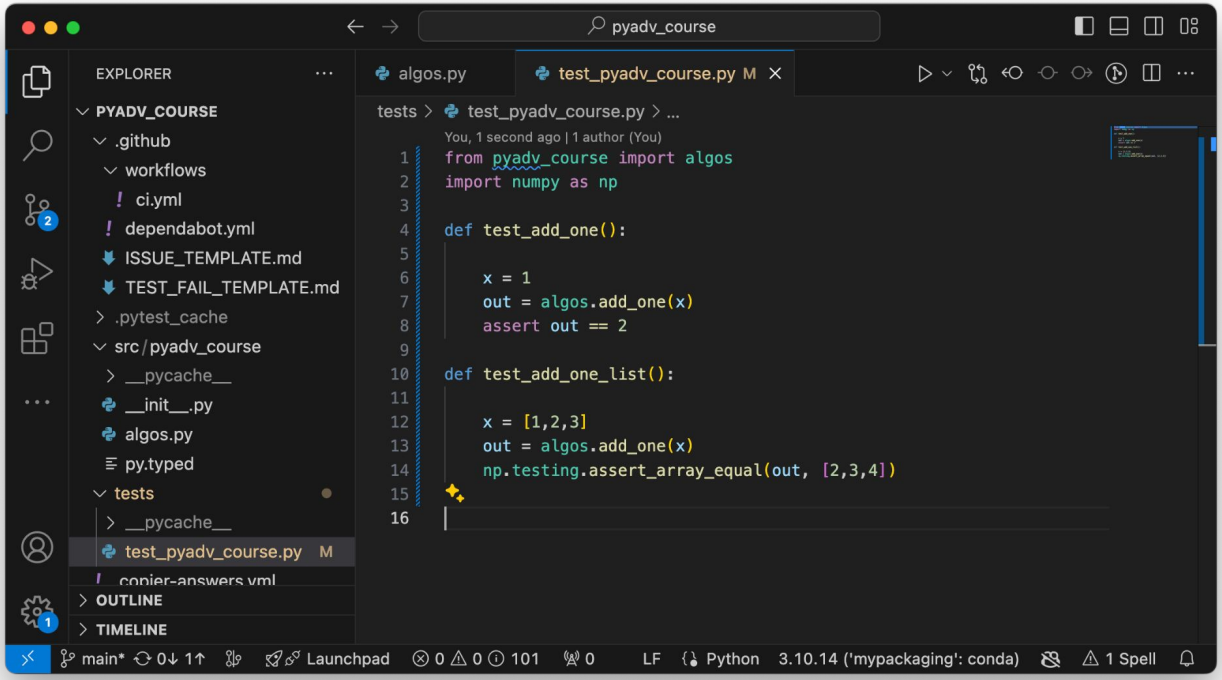
DSL

Tests for package, pytest

Adding tests

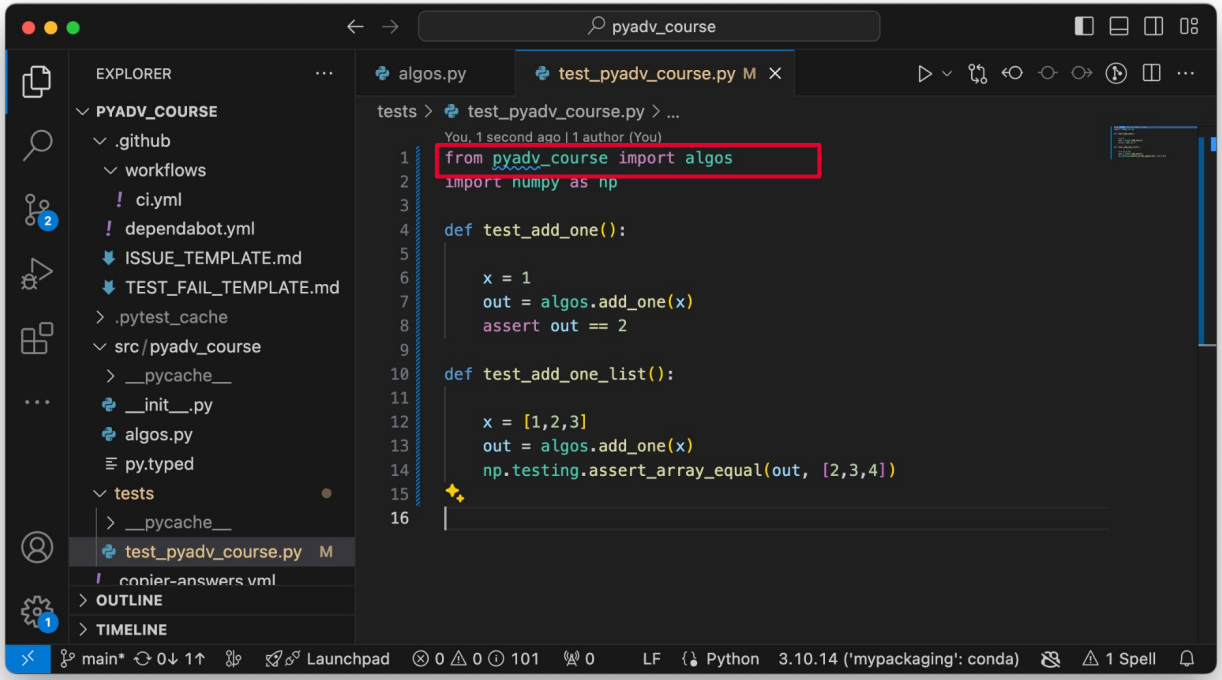
The package already has a test folder with a test module

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



Adding tests

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

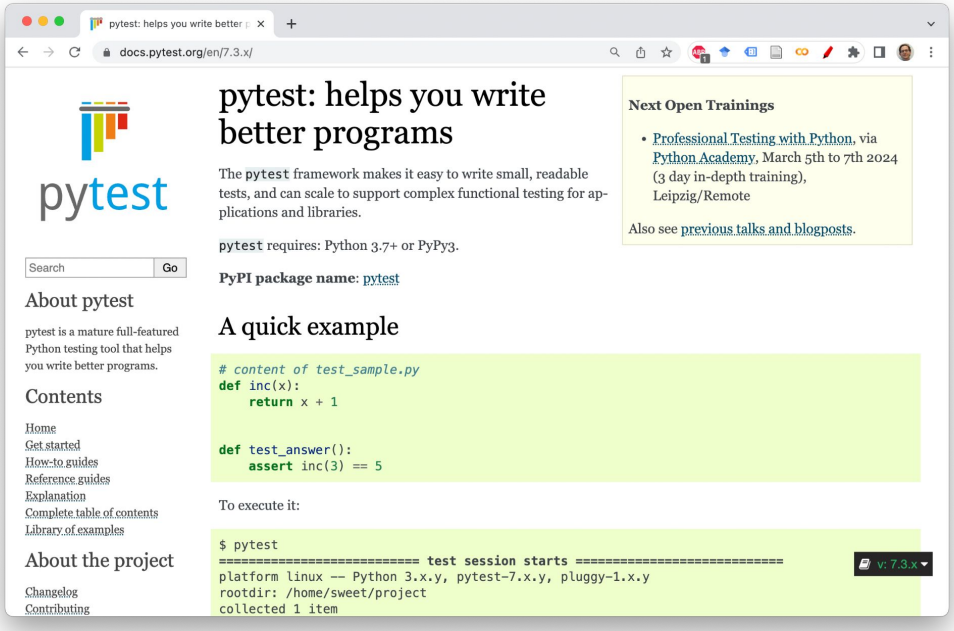


pytest

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



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

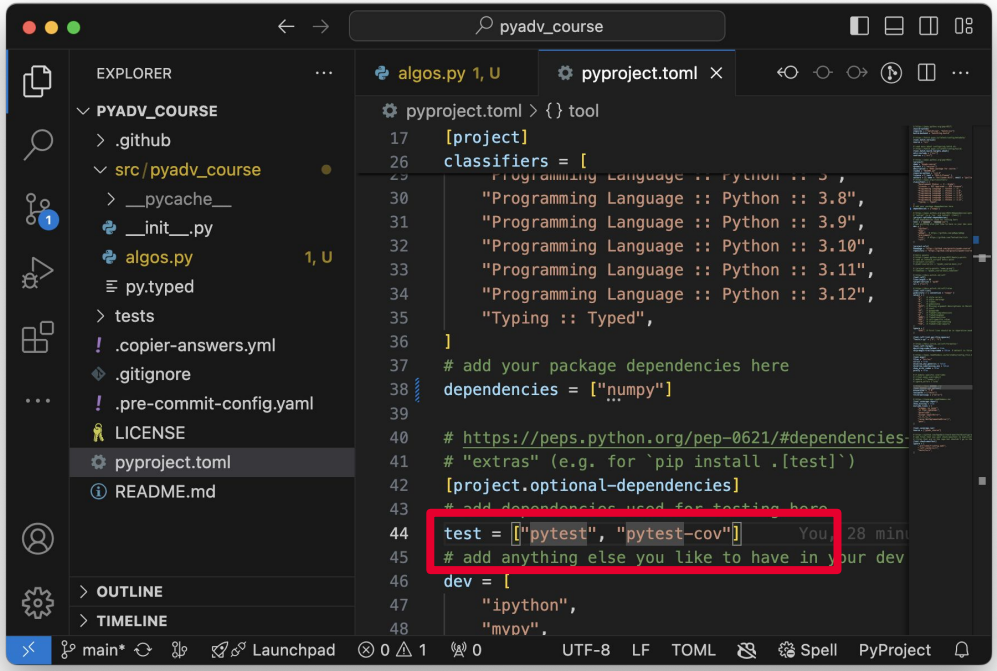
Run pytest

First, install in env with:

```
pip install pytest
```

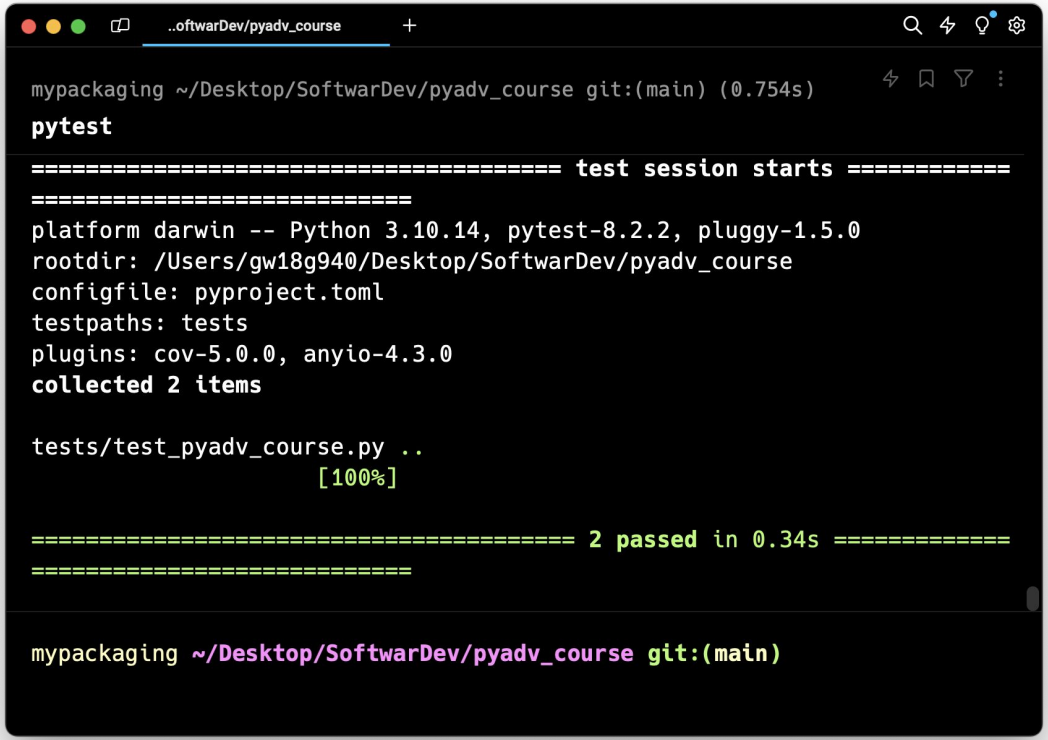
Or use your optional install!

```
pip install ".[test]"
```



Run pytest

Then from inside project folder: **`pytest`**



Test parametrization

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

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

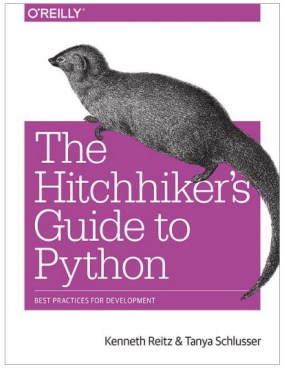
[1.1, 2.2] with expected results [2.1, 3.2]

```
@pytest.mark.parametrize("test_input, expected",
                         [([1, 2], [2, 3]), ([1.1, 2.2], [2.1, 3.2])])
def test_add_multiple(test_input, expected):

    out = algos.add_one(test_input)
    np.testing.assert_array_equal(out, expected)
```

General rules

O'Reilly Book

This guide is now available in tangible book form!



All proceeds are being directly donated to the DjangoGirls 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:

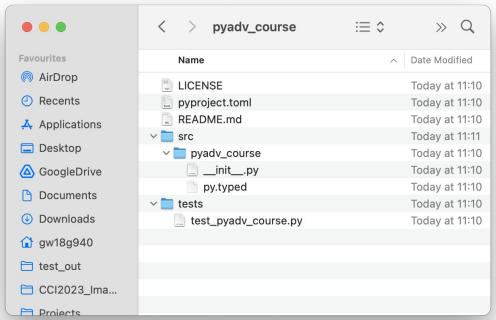
- A testing unit should focus on one tiny bit of functionality and prove it correct.
- Each test unit must be fully independent. Each test must be able to run alone, and also within the test suite, regardless of the order that they are called. The implication of this rule is that each test must be loaded 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.python-guide.org/writing/tests/

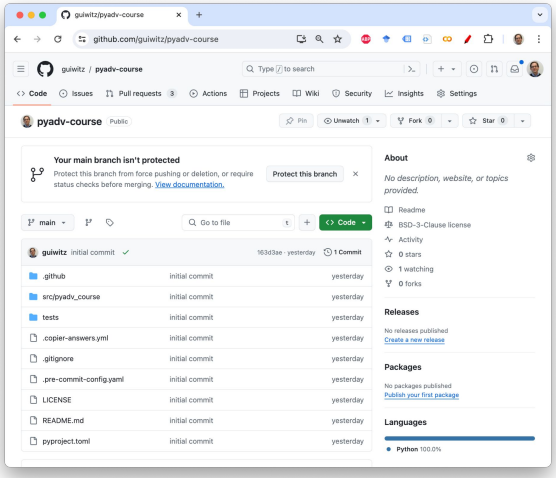
DSL

Running test automatically: GitHub actions

GitHub actions



push



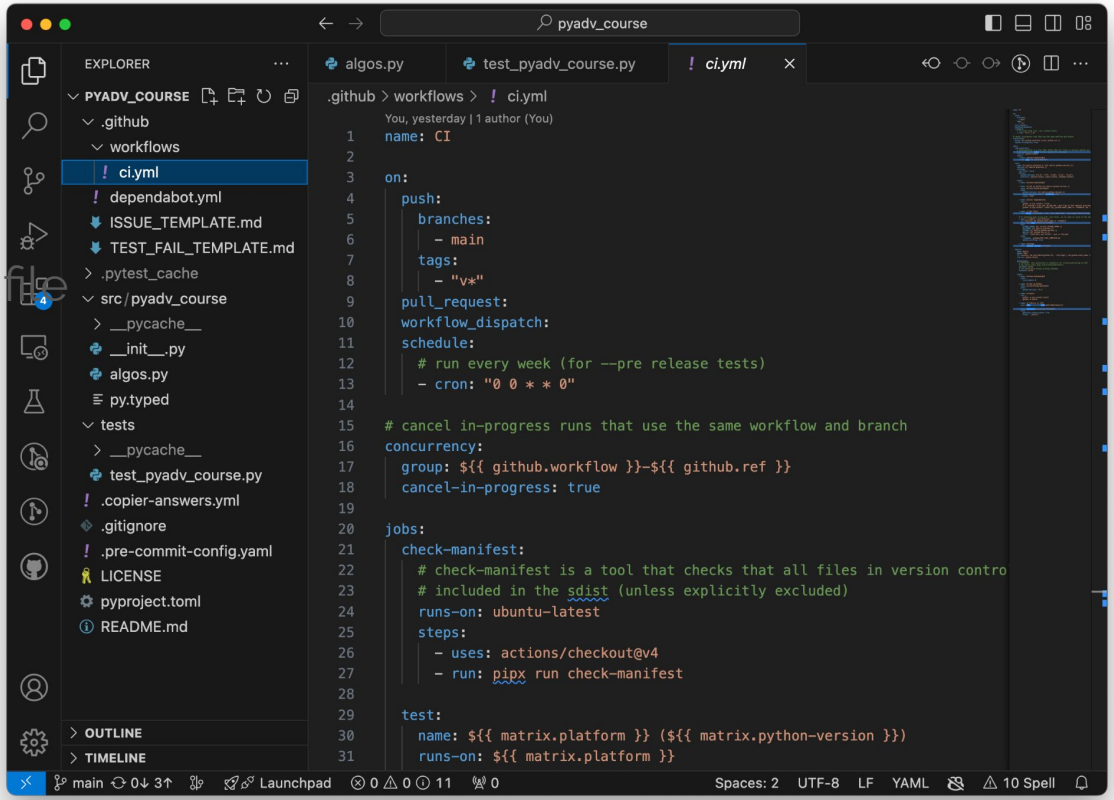
Run scripts upon a push

Actions specified in yml file

File create by copier:

.github/workflows/ci.yml

The file is hidden on the regular file system



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

```yaml
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 cont
    # 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

```yaml
jobs:
  check-manifest:
    # check-manifest is a tool that checks that all files in version control are
    # 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 }}
    strategy:
      fail-fast: false
      matrix:
        python-version: ["3.8", "3.9", "3.10", "3.11", "3.12"]
        platform: [ubuntu-latest, macos-latest, windows-latest]

    steps:
      - uses: actions/checkout@v4

      - name: 🐍 Set up Python ${{ matrix.python-version }}
        uses: actions/setup-python@v4
        with:
          python-version: ${{ matrix.python-version }}
          cache-dependency-path: "pyproject.toml"
          cache: "pip"

      - name: Install Dependencies
        run: |
          python -m pip install -U pip
          # if running a cron job, we add the --pre flag to test against pre-releases
          python -m pip install .[test] ${{ github.event_name == 'schedule' && '--pre' || '' }}

      - name: 🧪 Run Tests
        run: pytest --color=yes --cov --cov-report=xml --cov-report=term-missing

      # If something goes wrong with --pre tests, we can open an issue in the repo
      - name: 📝 Report --pre Failures
        if: failure() && github.event_name == 'schedule'
        uses: JasonEtco/create-an-issue@v2
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
          PLATFORM: ${{ matrix.platform }}
          PYTHON: ${{ matrix.python-version }}
          RUN_ID: ${{ github.run_id }}
          TITLE: "[test-bot] pip install --pre is failing"
        with:
          filename: .github/TEST_FAIL_TEMPLATE.md
          update_existing: true

      - name: Coverage
        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.

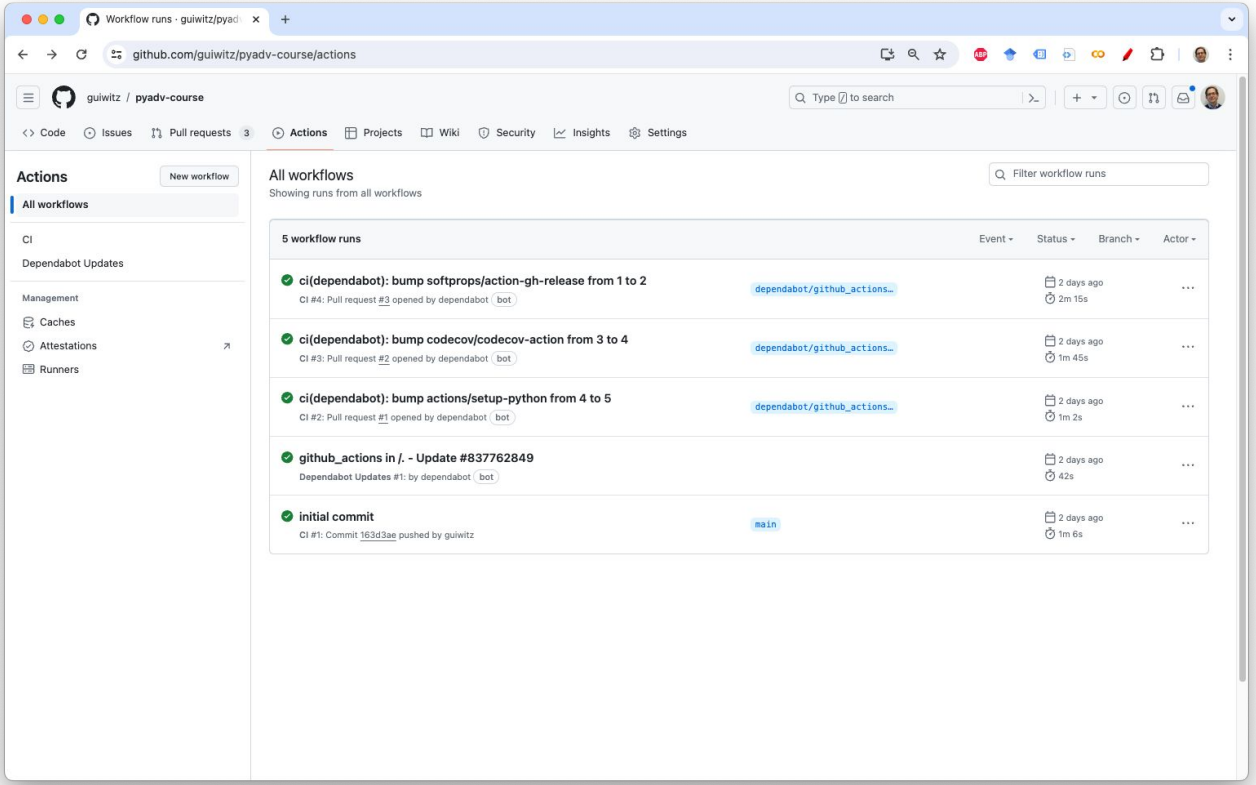
```yaml
jobs:
  check-manifest:
    # check-manifest is a tool that checks that all files in version control are
    # included in the sdist (unless explicitly excluded)
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - run: pipx run check-manifest

  test:
    name: ${{ matrix.platform }} (${{ matrix.python-version }})
    runs-on: ${{ matrix.platform }}
    strategy:
      fail-fast: false
      matrix:
        python-version: ["3.8", "3.9", "3.10", "3.11", "3.12"]
        platform: [ubuntu-latest, macos-latest, windows-latest]

    steps:
      - uses: actions/checkout@v4

      - name: 🐍 Set up Python ${{ matrix.python-version }}
        uses: actions/setup-python@v4
        with:
          python-version: ${{ matrix.python-version }}
          cache-dependency-path: "pyproject.toml"
          cache: "pip"

      - name: Install Dependencies
        run: |
          python -m pip install -U pip
          # if running a cron job, we add the --pre flag to test against pre-releases
          python -m pip install .[test] ${{ github.event_name == 'schedule' && '--pre' || '' }}

      - name: 🧪 Run Tests
        run: pytest --color=yes --cov --cov-report=xml --cov-report=term-missing

      # If something goes wrong with --pre tests, we can open an issue in the repo
      - name: 📝 Report --pre Failures
        if: failure() && github.event_name == 'schedule'
        uses: JasonEtco/create-an-issue@v2
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
          PLATFORM: ${{ matrix.platform }}
          PYTHON: ${{ matrix.python-version }}
          RUN_ID: ${{ github.run_id }}
          TITLE: "[test-bot] pip install --pre is failing"
        with:
          filename: .github/TEST_FAIL_TEMPLATE.md
          update_existing: true

      - name: Coverage
        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

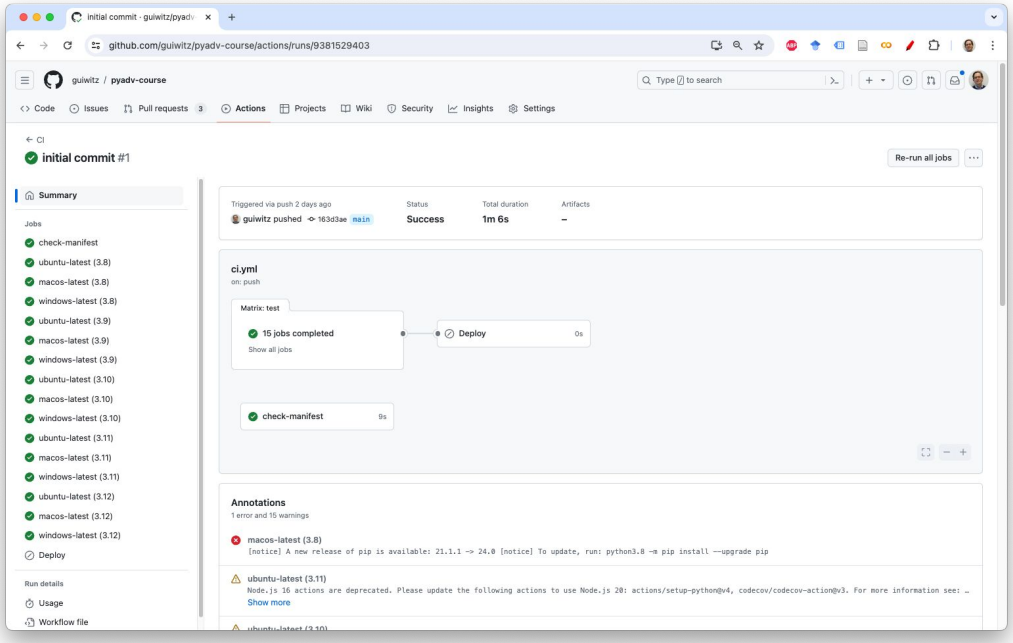


Actions specified in yml file

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/pyadvcourse/actions/runs/5215870500/jobs/9413950714

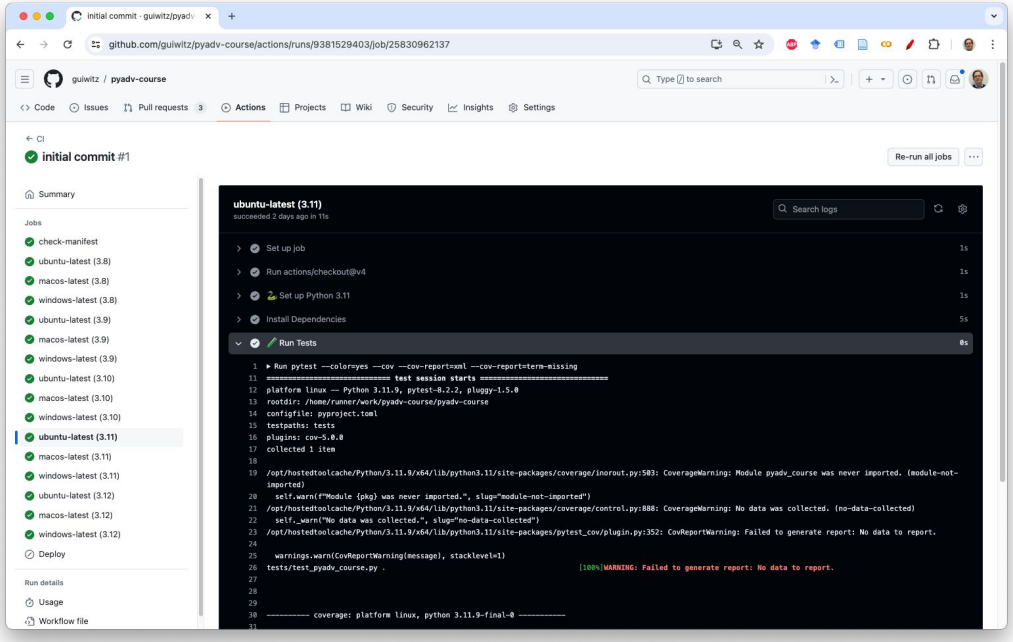
Actions specified in yml file

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



DSL

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

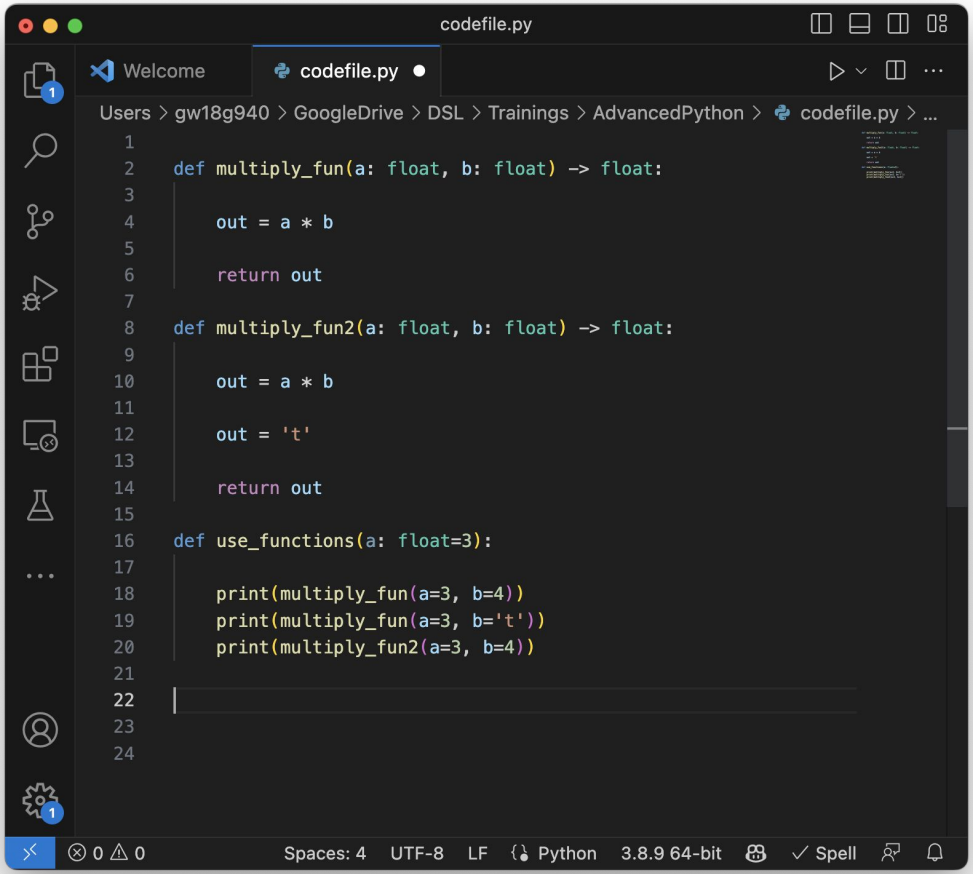
Tools like mypy go through code and check for type inconsistencies

Here we have:

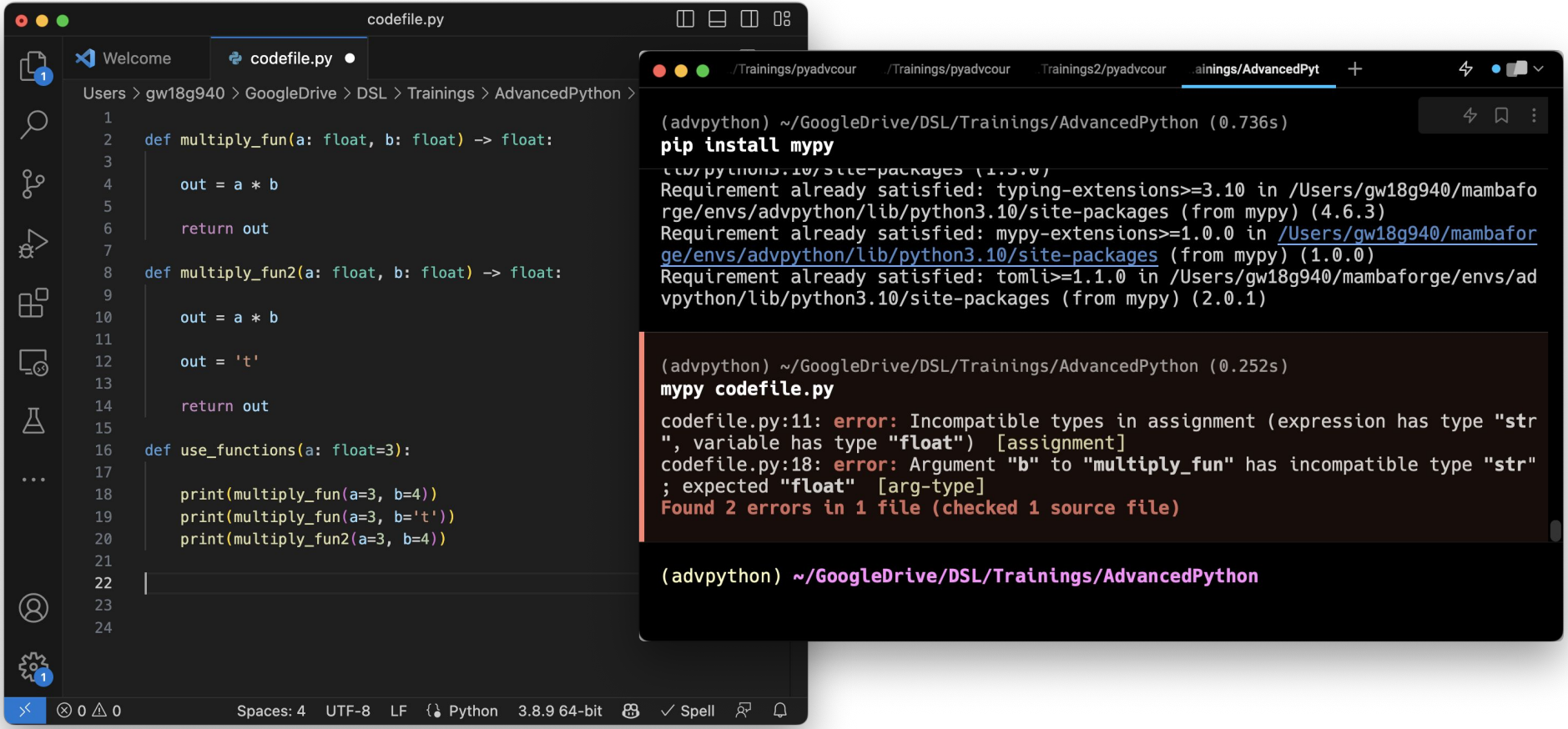
- bad inputs in `use_functions`
- bad output in `multiply_fun2`

Install mypi to find those issues:

```
pip install mypy
```



mypi


codefile.py
Welcome
codefile.py
Users > gw18g940 > GoogleDrive > DSL > Trainings > AdvancedPython >
1 def multiply_fun(a: float, b: float) -> float:
2
3 out = a * b
4
5 return out
6
7 def multiply_fun2(a: float, b: float) -> float:
8
9 out = a * b
10
11 out = 't'
12
13 return out
14
15 def use_functions(a: float=3):
16
17 print(multiply_fun(a=3, b=4))
18 print(multiply_fun(a=3, b='t'))
19 print(multiply_fun2(a=3, b=4))
20
21
22
23
24
0 0
Spaces: 4 UTF-8 LF Python 3.8.9 64-bit Spell
/Trainings/pyadvcour
/Trainings/pyadvcour
Trainings2/pyadvcour
ainings/AdvancedPyt
(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

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

```yaml
name: CI

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

Deploy job

Only works:

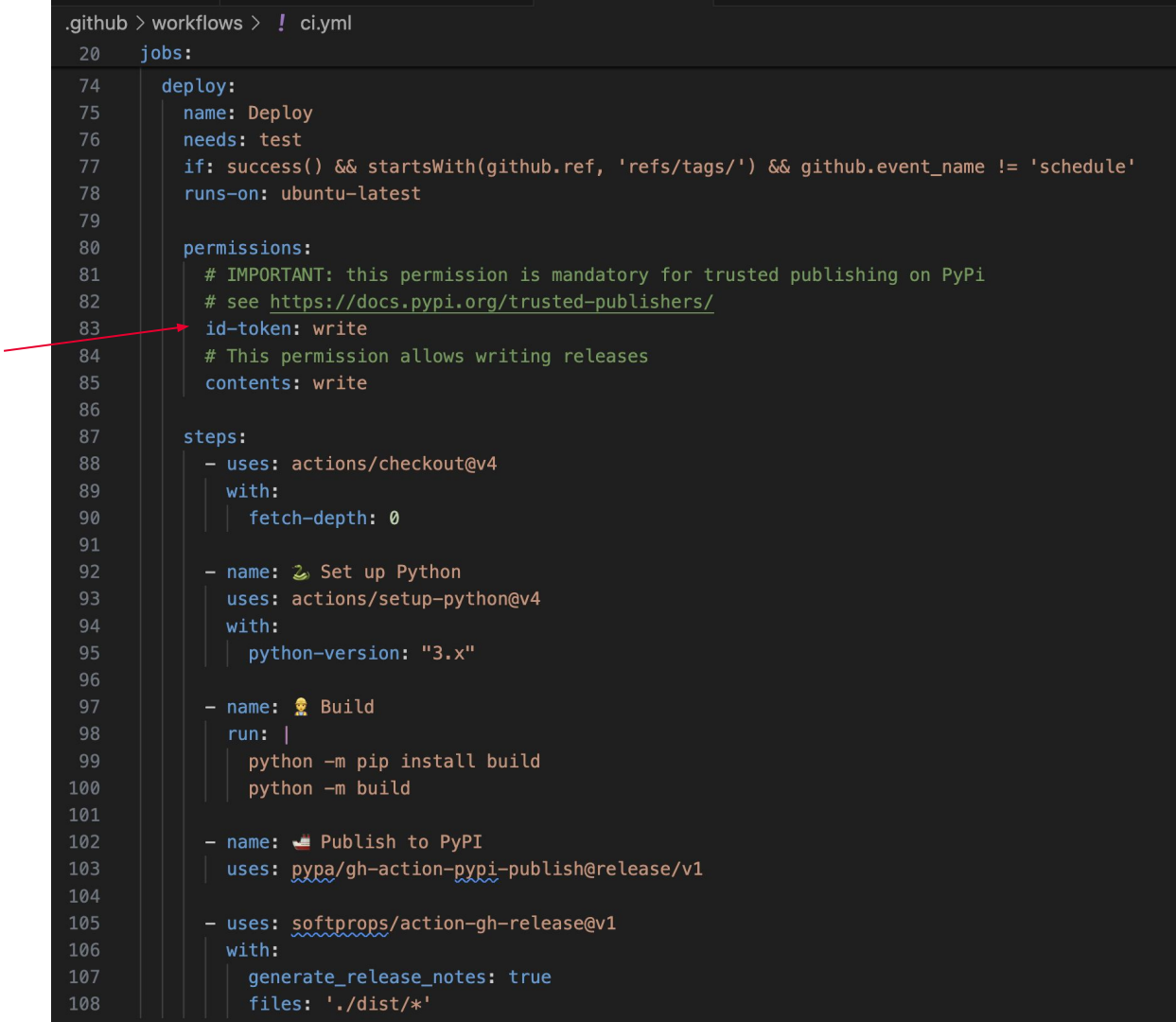
- if tests are run
- if previous jobs are successful
  - if the commit was tagged
  - if it's not a scheduled run

.github > workflows > ! ci.yml

```yaml
jobs:
  deploy:
    name: Deploy
    needs: test
    if: success() && startsWith(github.ref, 'refs/tags/') && github.event_name != 'schedule'
    runs-on: ubuntu-latest

    permissions:
      # IMPORTANT: this permission is mandatory for trusted publishing on PyPi
      # see https://docs.pypi.org/trusted-publishers/
      id-token: write
      # This permission allows writing releases
      contents: write

    steps:
      - uses: actions/checkout@v4
        with:
          fetch-depth: 0

      - name: 🐍 Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: "3.x"

      - name: 👷 Build
        run: |
          python -m pip install build
          python -m build

      - name: 🚢 Publish to PyPI
        uses: pypa/gh-action-pypi-publish@release/v1

      - uses: softprops/action-gh-release@v1
        with:
          generate_release_notes: true
          files: './dist/*'
```

Deploy job

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



.github > workflows > ! ci.yml

```yaml
jobs:
  deploy:
    name: Deploy
    needs: test
    if: success() && startsWith(github.ref, 'refs/tags/') && github.event_name != 'schedule'
    runs-on: ubuntu-latest

    permissions:
      # IMPORTANT: this permission is mandatory for trusted publishing on PyPi
      # see https://docs.pypi.org/trusted-publishers/
      id-token: write
      # This permission allows writing releases
      contents: write

    steps:
      - uses: actions/checkout@v4
        with:
          fetch-depth: 0

      - name: 🐍 Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: "3.x"

      - name: 👷 Build
        run: |
          python -m pip install build
          python -m build

      - name: 🚢 Publish to PyPI
        uses: pypa/gh-action-pypi-publish@release/v1

      - uses: softprops/action-gh-release@v1
        with:
          generate_release_notes: true
          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

```yaml
jobs:
  deploy:
    name: Deploy
    needs: test
    if: success() && startsWith(github.ref, 'refs/tags/') && github.event_name != 'schedule'
    runs-on: ubuntu-latest

    permissions:
      # IMPORTANT: this permission is mandatory for trusted publishing on PyPi
      # see https://docs.pypi.org/trusted-publishers/
      id-token: write
      # This permission allows writing releases
      contents: write

    steps:
      - uses: actions/checkout@v4
        with:
          fetch-depth: 0

      - name: 🐍 Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: "3.x"

      - name: 👷 Build
        run: |
          python -m pip install build
          python -m build

      - name: 🚢 Publish to PyPI
        uses: pypa/gh-action-pypi-publish@release/v1

      - uses: softprops/action-gh-release@v1
        with:
          generate_release_notes: true
          files: './dist/*'
```

TestPyPi

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

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

Please add the following in the Publish section:

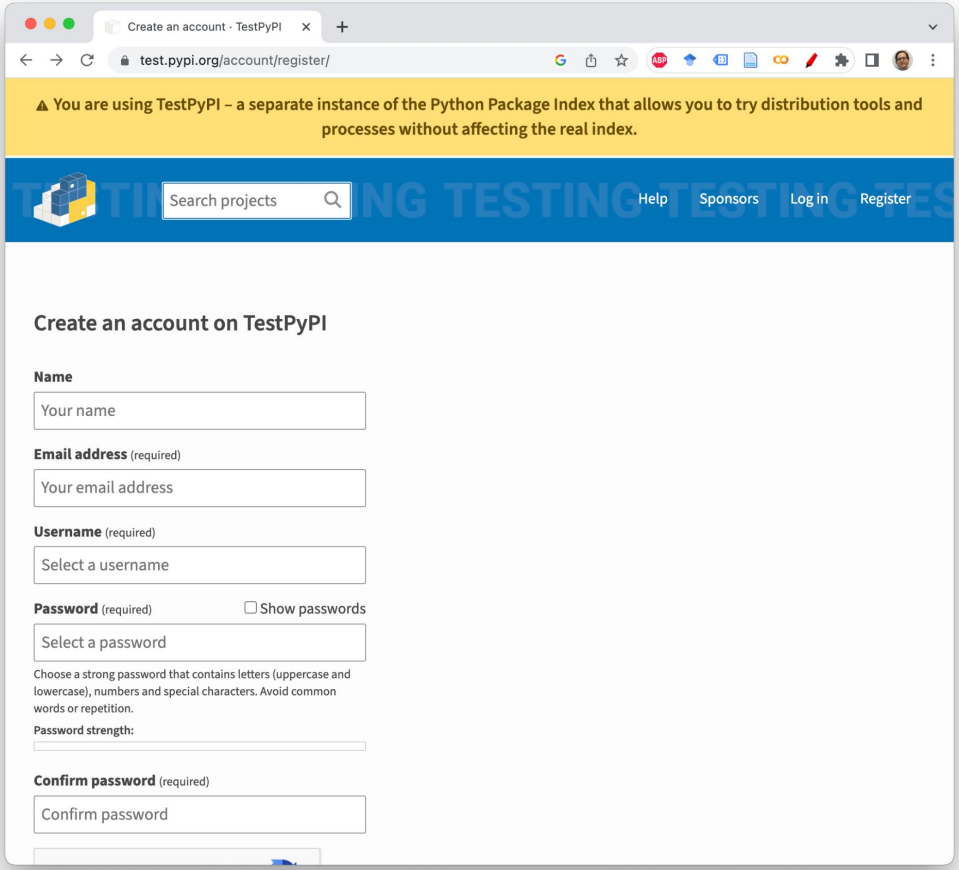
```yaml
with:
    repository-url: https://test.pypi.org/legacy/
```

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

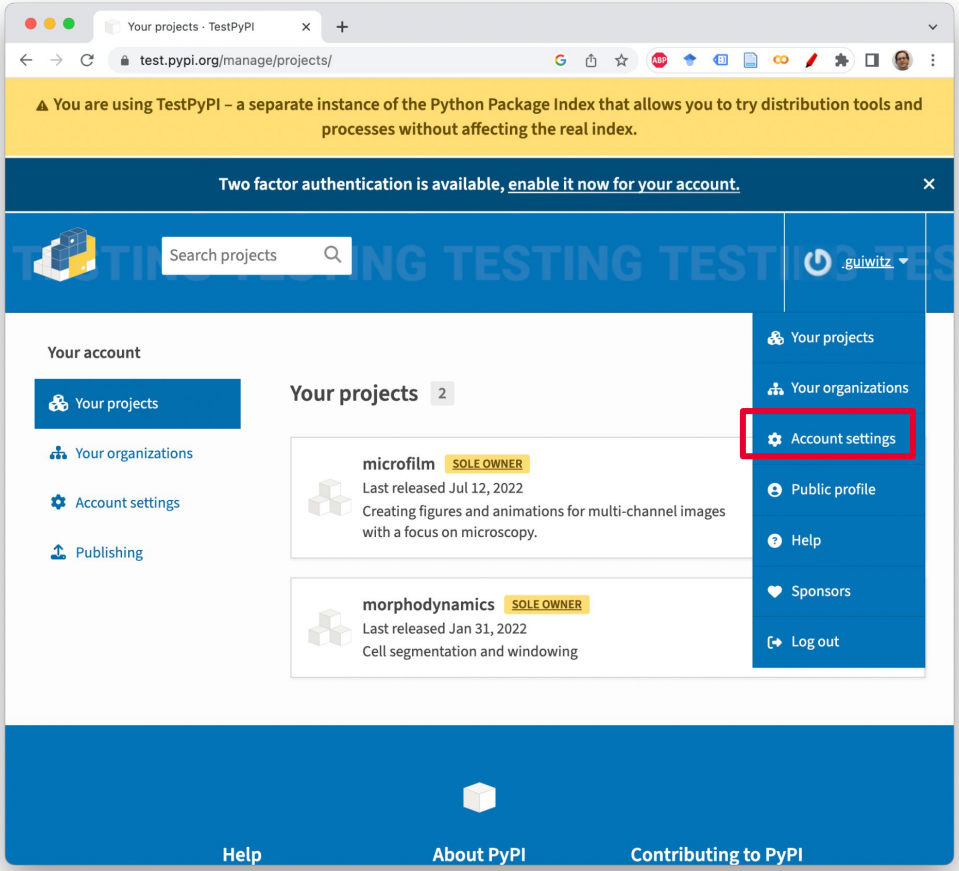
https://test.pypi.org/account/register/



Create account on TestPyPi

Login

Go to account settings



Create account on TestPyPi

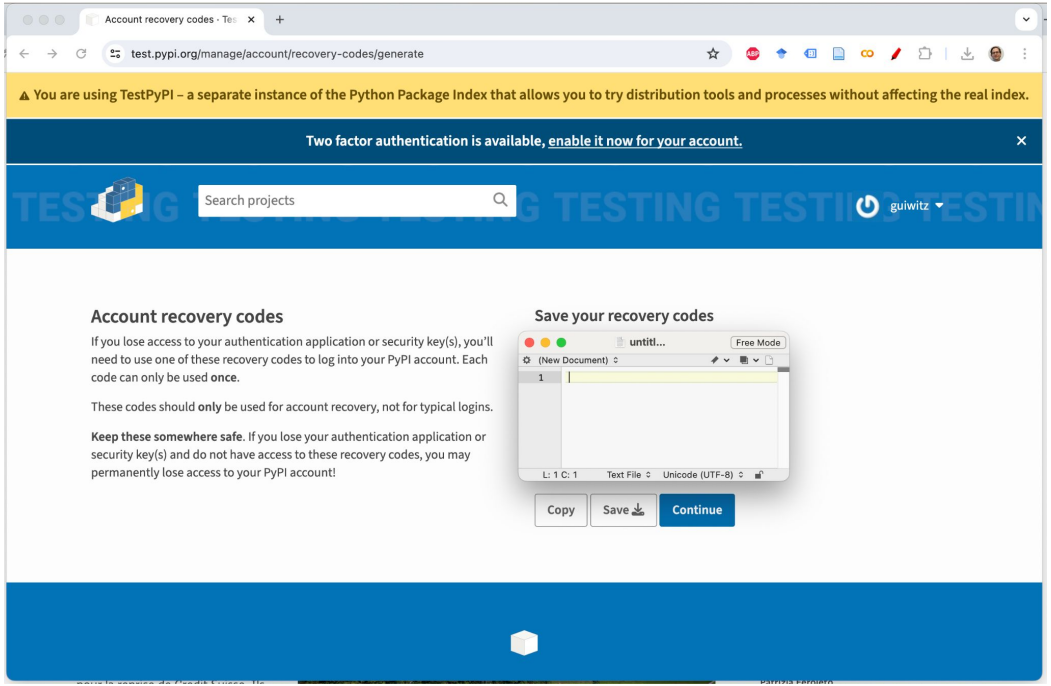
Login

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

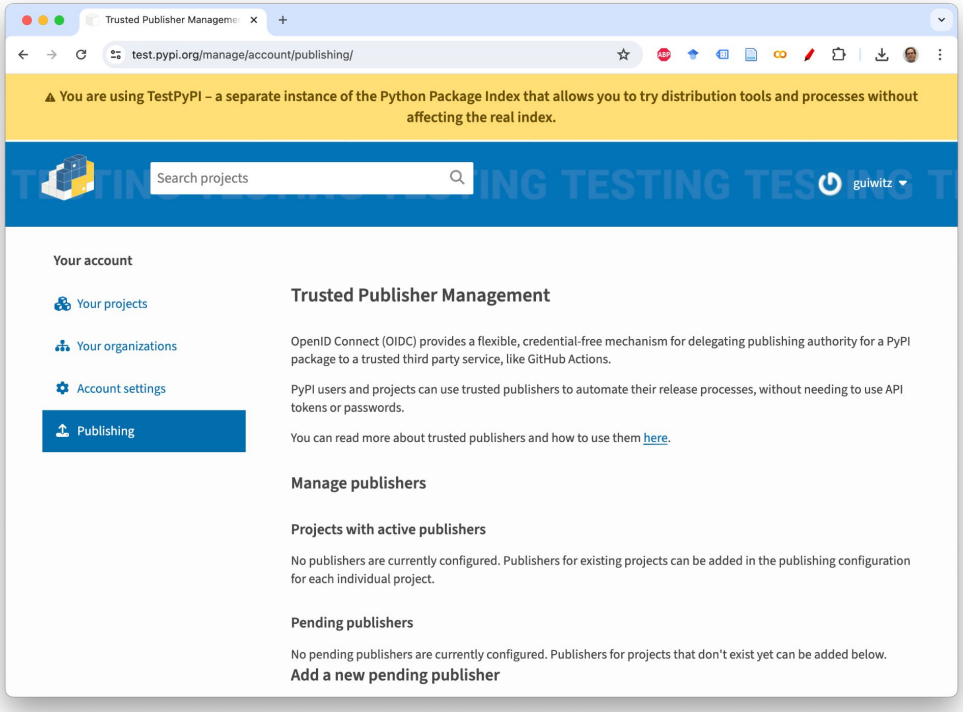
Go to account settings

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

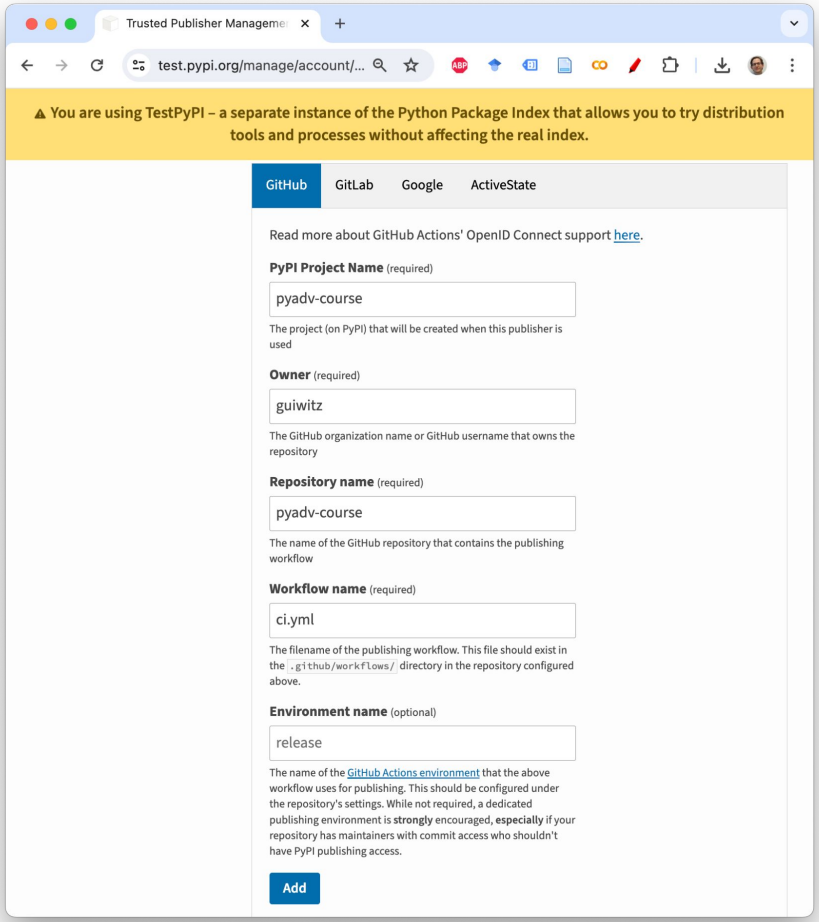
Enable two-factor authentication:

1. Save recovery codes

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

Go to Publishing tab

Fill the form



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:

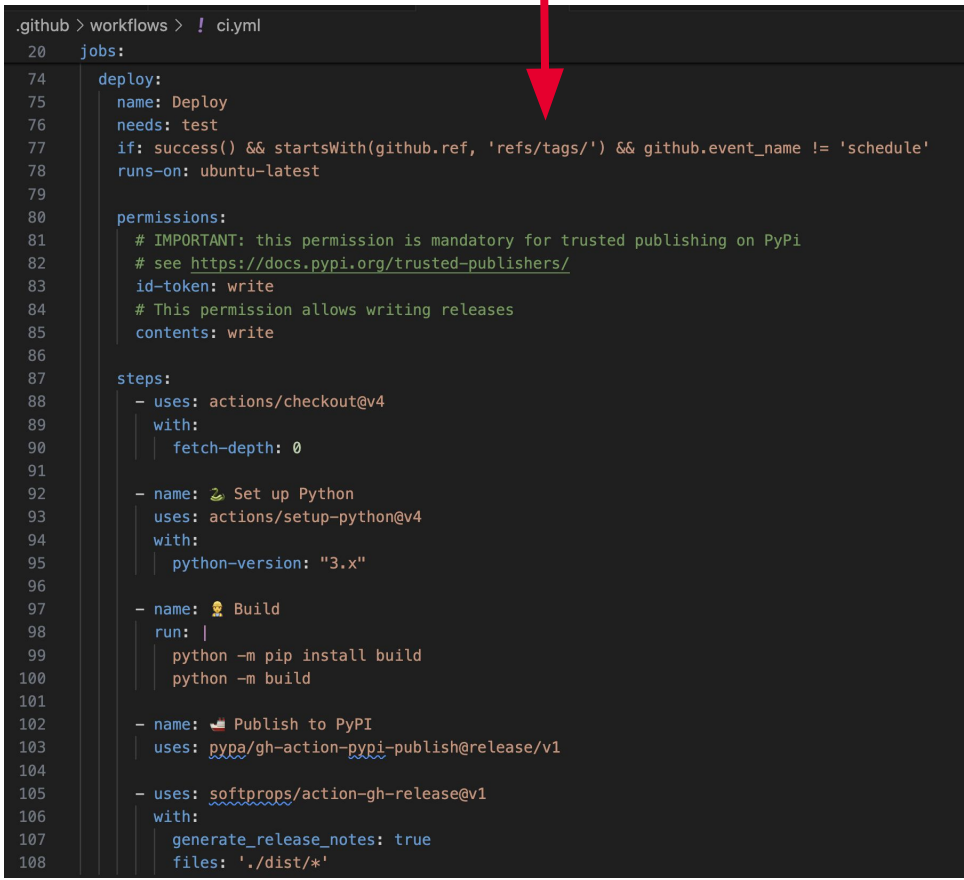
Add a new release tag starting with v:

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

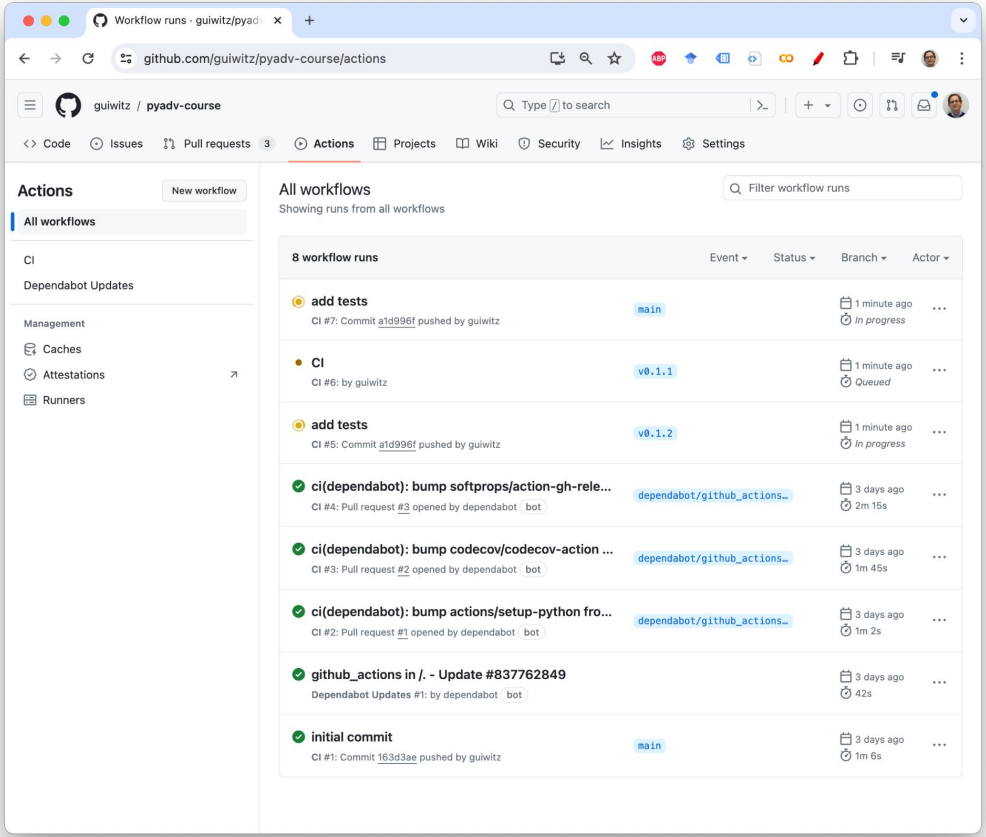
Push everything from VSCode

Push your tag:

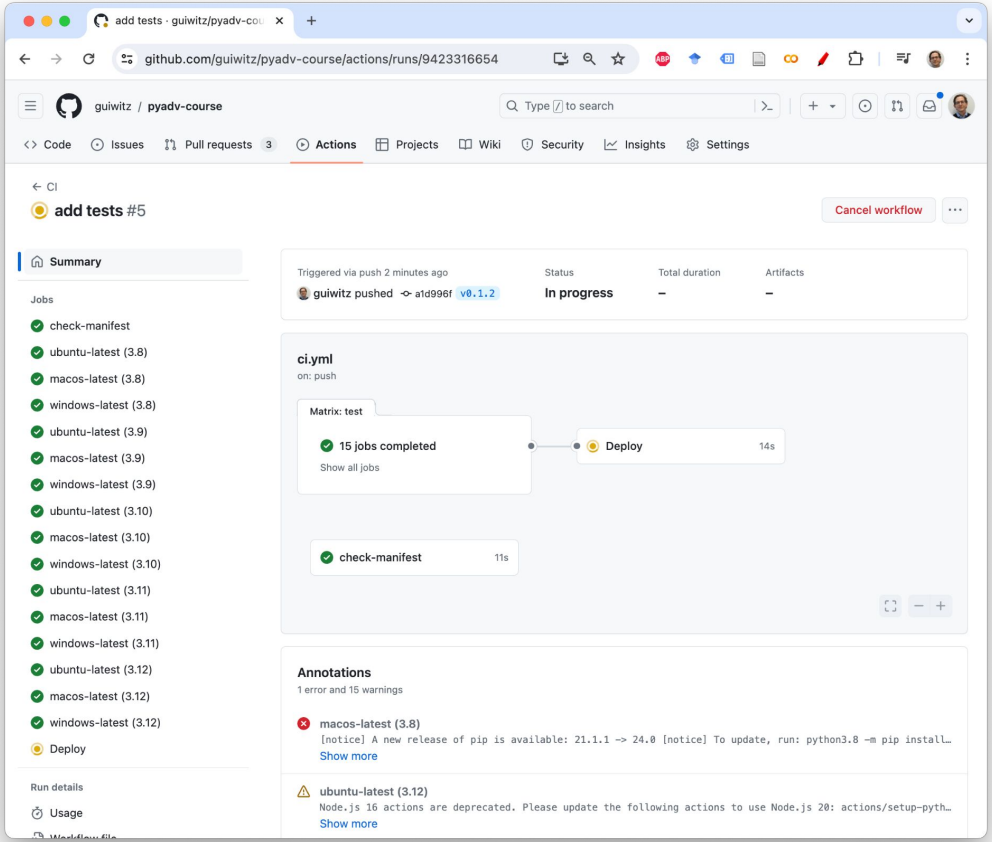
```
git push --follow-tags
```



Finally we can try to release

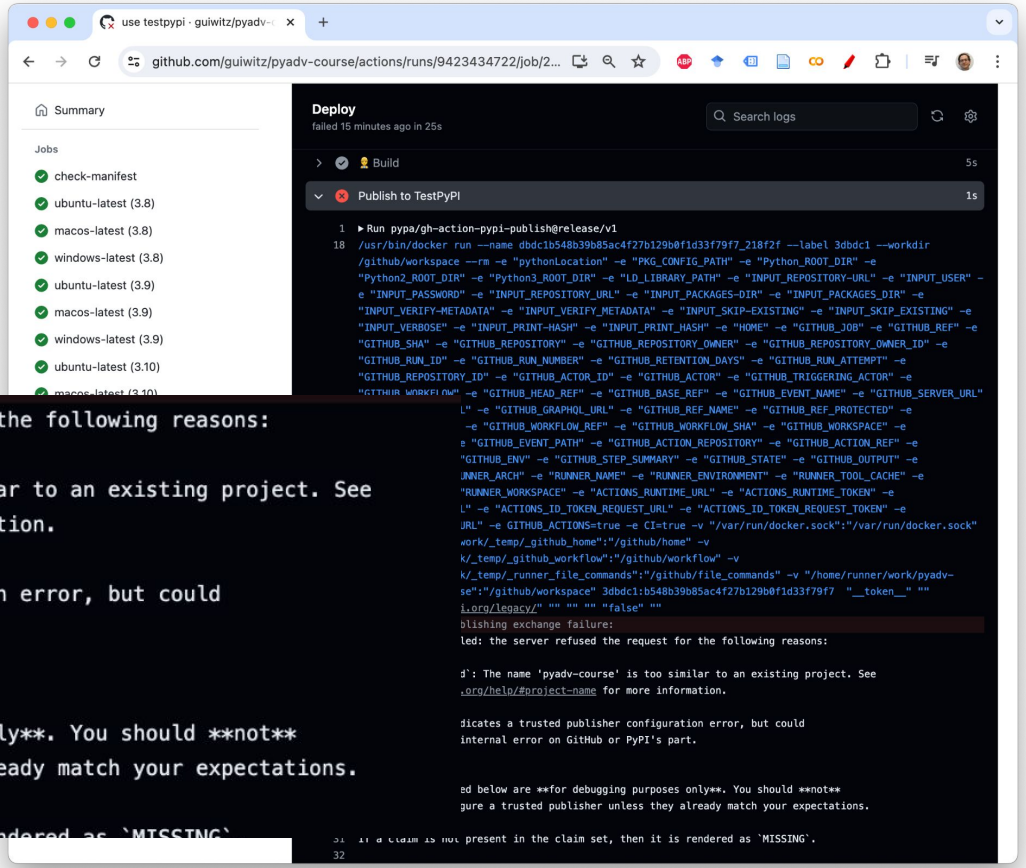


Finally we can try to release

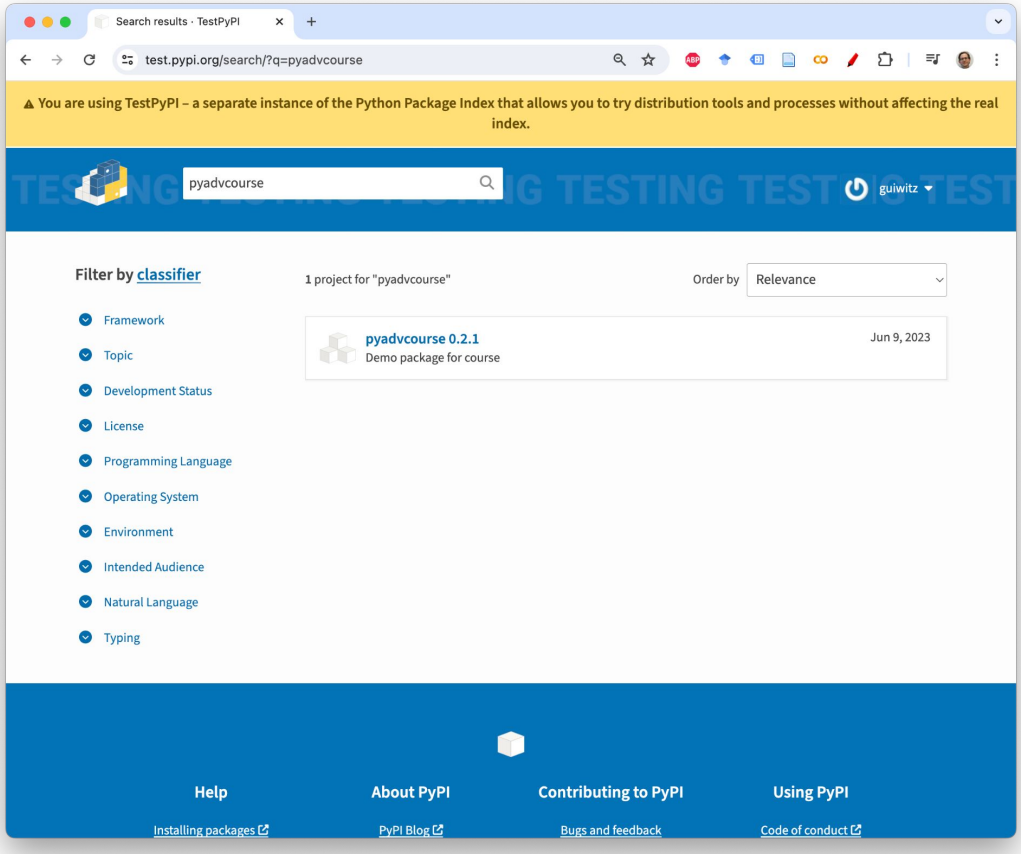


Finally we can try to release… and fail

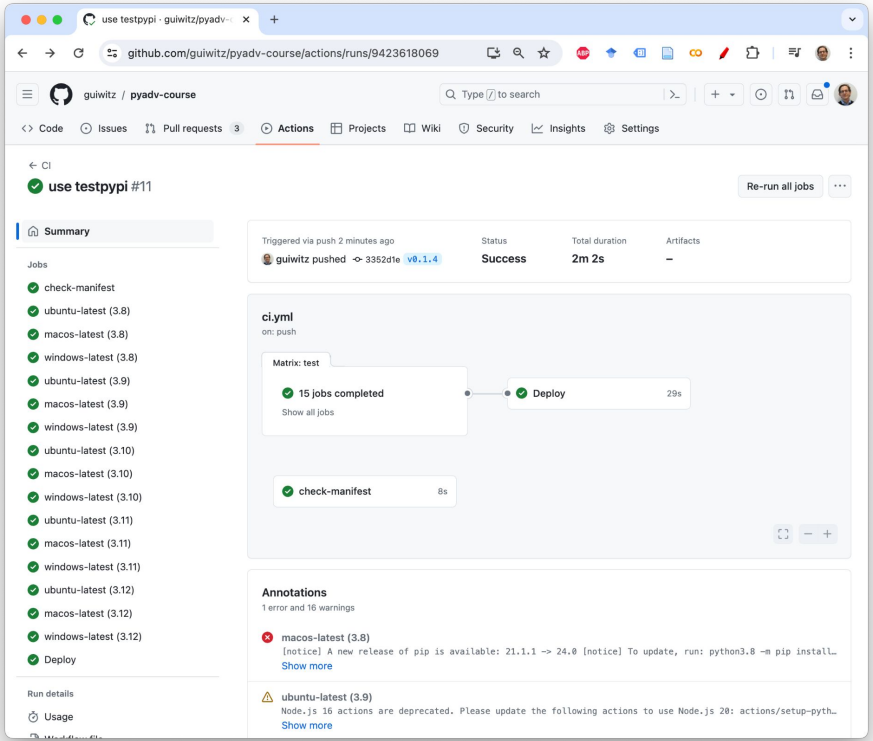
Name of package is already taken
or close to a taken one!

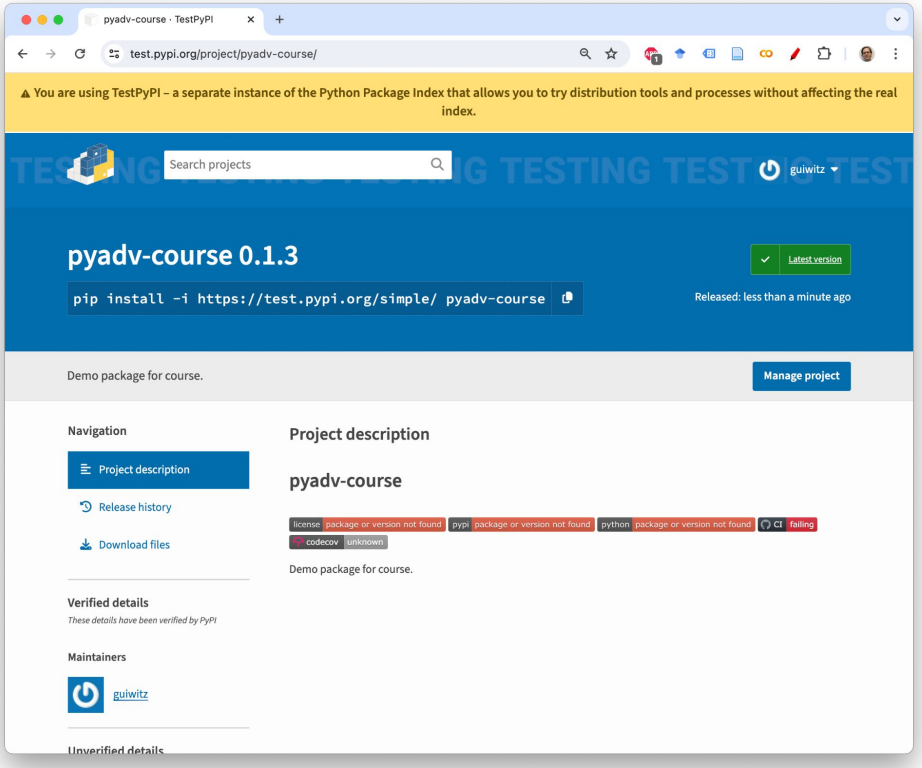


Finally we can try to release… and fail



Houraaaaa


use testpypi · guiwitz/pyadv-
github.com/guiwitz/pyadv-course/actions/runs/9423618069
guiwitz / pyadv-course
Type / to search
Code
Issues
Pull requests 3
Actions
Projects
Wiki
Security
Insights
Settings
CI
use testpypi #11
Re-run all jobs
Summary
Jobs
check-manifest
ubuntu-latest (3.8)
macos-latest (3.8)
windows-latest (3.8)
ubuntu-latest (3.9)
macos-latest (3.9)
windows-latest (3.9)
ubuntu-latest (3.10)
macos-latest (3.10)
windows-latest (3.10)
ubuntu-latest (3.11)
macos-latest (3.11)
windows-latest (3.11)
ubuntu-latest (3.12)
macos-latest (3.12)
windows-latest (3.12)
Deploy
Run details
Usage
Triggered via push 2 minutes ago
guiwitz pushed 3352d1e v0.1.4
Status
Success
Total duration
2m 2s
Artifacts
–
ci.yml
on: push
Matrix: test
15 jobs completed
Show all jobs
Deploy
29s
check-manifest
8s
Annotations
1 error and 16 warnings
macos-latest (3.8)
[notice] A new release of pip is available: 21.1.1 -> 24.0 [notice] To update, run: python3.8 -m pip install…
Show more
ubuntu-latest (3.9)
Node.js 16 actions are deprecated. Please update the following actions to use Node.js 20: actions/setup-pyth…
Show more


pyadv-course · TestPyPI
test.pypi.org/project/pyadv-course/
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.
Search projects
guiwitz
pyadv-course 0.1.3
pip install -i https://test.pypi.org/simple/ pyadv-course
Latest version
Released: less than a minute ago
Demo package for course.
Manage project
Navigation
Project description
Release history
Download files
Verified details
These details have been verified by PyPI
Maintainers
guiwitz
Unverified details
Project description
pyadv-course
license package or version not found
pypi package or version not found
python package or version not found
CI failing
codecov unknown
Demo package for course.