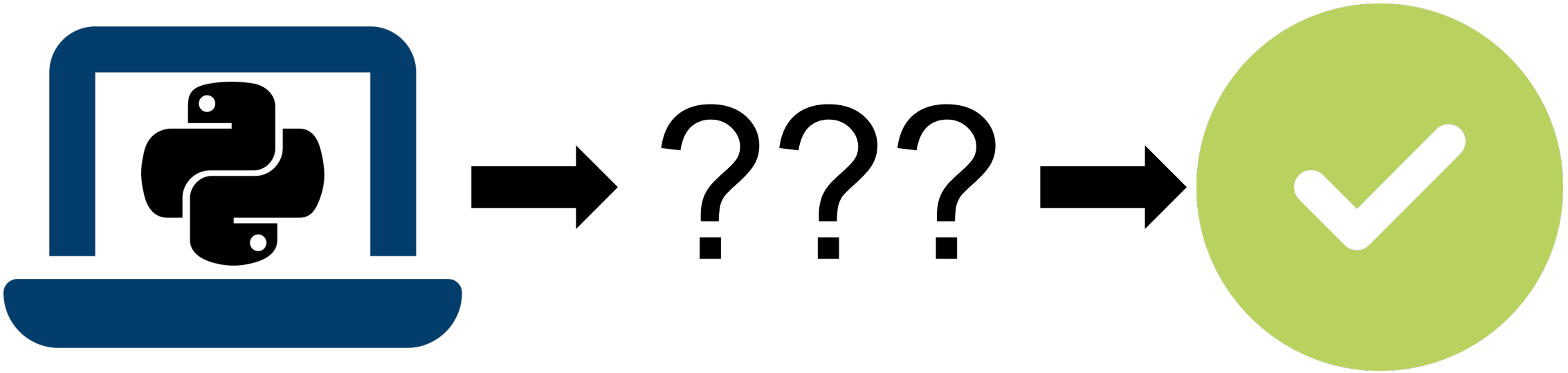


TESTING IN PYTHON

An Introduction and Getting Started

SUMMER 2023 I JAKOB FRITZ I HIRSE SUMMER OF TESTING

OVERVIEW

- Linting & Formatting
- How to do unit-tests with pytest
- Test coverage
- Property based testing
- Conclusion & Further reading

OVERVIEW

- Linting & Formatting
- How to do unit-tests with pytest
- Test coverage
- Property based testing
- Conclusion & Further reading

LINTING & FORMATTING

Linting

- Easy way to check for syntactic correctness
- No check on semantic correctness
- E.g. Flake8, pylint, ruff
- Find missing brackets, unused imports, unused variables, to long lines and more

HiRSE_SoT/linting.py

```python
# Import an unused package
import pandas as pd

def exciting_function():
    print("This function can print text")
    print("It can also print very long texts that should be split into multiple lines to comply with standards")
```

```
$ ruff .
HiRSE_SoT/linting.py:7:18: F401 [*] `pandas` imported but unused
HiRSE_SoT/linting.py:11:89: E501 Line too long (112 > 88 characters)
Found 2 errors.
[*] 1 potentially fixable with the --fix option.
```

LINTING & FORMATTING

Linting

- Easy way to check for syntactic correctness
- No check on semantic correctness
- E.g. Flake8, pylint, ruff
- Find missing brackets, unused imports, unused variables, to long lines and more

After manual reformatting

HiRSE_SoT/linting.py

```
# Import an unused package
# import pandas as pd

def exciting_function():
    print("This function can print text")
    print("It can also print very long texts "
          +"that should be split into multiple lines to comply with standards")
```

```
$ ruff .
$
```

LINTING & FORMATTING

(Auto) Formatting

- Consistent style of code (across developers & time)
- Increase readability of code
- E.g. black

HiRSE_SoT/linting.py

```python
# Import an unused package
import pandas as pd

def exciting_function():
    print("This function can print text")
    print("It can also print very long texts that should be split into multiple lines to comply with standards
```

```
$ black .
reformatted ./HiRSE_SoT/linting.py

All done! ✨ 🍰 ✨
1 file reformatted, 0 files left unchanged.
```

LINTING & FORMATTING

(Auto) Formatting

After automatic reformatting

HiRSE_SoT/linting.py

```python
# Import an unused package
import pandas as pd


def exciting_function():
    print("This function can print text")
    print(
        "It can also print very long texts that should be split into multiple lines to comply with standards”
    )
```

Two empty lines between functions

Additional linebreaks for shorter lines

```
$ black .

All done! ✨ 🍰 ✨
0 files reformatted, 1 file left unchanged.
```

LINTING & FORMATTING

Combining Linting & Formatting

- Both have strengths
  - Linting finds unused packages
  - Autoformatting takes care of e.g. line lengths
- Often good idea to combine them
- My current approach:
  - Write Code
  - Run black & ruff
  - Take care of errors from ruff
  - Run black & ruff again (hopefully now unchanged)

LINTING & FORMATTING

(Auto) Formatting

Before automatic & manual reformatting

HiRSE_SoT/linting.py

```python
# Import an unused package
import pandas as pd

def exciting_function():
    print("This function can print text")
    print("It can also print very long texts that should be split into multiple lines to comply with standards'
```

```
$ black .
reformatted ./HiRSE_SoT/linting.py

All done! ✨ 🍰 ✨
1 file reformatted, 0 files left unchanged.
$ ruff .
HiRSE_SoT/linting.py:6:18: F401 [*] `pandas` imported but unused
HiRSE_SoT/linting.py:12:89: E501 Line too long (109 > 88 characters)
Found 2 errors.
[*] 1 potentially fixable with the --fix option.
```

LINTING & FORMATTING

(Auto) Formatting

After automatic & manual reformatting

HiRSE_SoT/linting.py

```python
# Import an unused package
# import pandas as pd


def exciting_function():
    print("This function can print text")
    print(
        "It can also print very long texts "
        + "that should be split into multiple lines to comply with standards"
    )
```

```
$ black .
All done! ✨ 🍰 ✨
1 file left unchanged.
$ ruff .
$
```

OVERVIEW

- Linting & Formatting
- How to do unit-tests with pytest
- Test coverage
- Property based testing
- Conclusion & Further reading

UNIT TESTS

- Checking for semantic correctness
- No check for performance of code
- E.g. pytest
- Need to specify tests and results

- Put tests in separate directory
- Name files "`test_*.py`" or "`*_test.py`"
- Prefix functions with "`test`"
- `__init__.py` files can be empty

Recommended directory Layout (adapted from [1]):

```
setup.py
mypkg/
    __init__.py
    app.py
    helper.py
tests/
    __init__.py
    test_app.py
    test_helper.py
```

[1] https://docs.pytest.org/en/latest/explanation/goodpractices.html

UNIT TESTS

First test

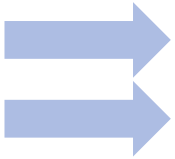
- Checking for semantic correctness
- No check for performance of code
- E.g. pytest
- Need to specify tests and results

HiRSE_SoT/example_funcs.py

```python
def is_even(number):
    modulo = number % 2
    if modulo == 0:
        return True
    else:
        return False
```

test/test_even.py

```python
from HiRSE_SoT import example_funcs


def test_is_even():
    result_4 = example_funcs.is_even(4)
    assert result_4
    result_5 = example_funcs.is_even(5)
    assert not result_5
```



Statements after `assert` must be true

UNIT TESTS

Run a test

HiRSE_SoT/example_funcs.py

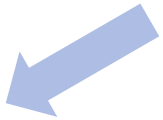
```python
def is_even(number):
    modulo = number % 2
    if modulo == 0:
        return True
    else:
        return False
```

test/test_even.py

```python
from HiRSE_SoT import example_funcs


def test_is_even():
    result_4 = example_funcs.is_even(4)
    assert result_4
    result_5 = example_funcs.is_even(5)
    assert not result_5
```

- Run `pip install –e .` to locally install your code (in editable mode)

```
$ pytest .
================ test session starts ================
collected 1 item

test/test_even.py .                                [100%]

================= 1 passed in 0.02s =================
```



One test ran and succeeded

UNIT TESTS

Check if errors are raised

HiRSE_SoT/example_funcs.py

```
…

def is_even_advanced(number):
    if not isinstance(number, int):
        raise TypeError("Expected an integer")
    return is_even(number=number)
```

test/test_even.py

```
import pytest
from HiRSE_SoT import example_funcs


def test_is_even():
…

def test_is_even_advanced():
    with pytest.raises(TypeError):
        example_funcs.is_even_advanced("NotAnInteger")
    with pytest.raises(TypeError):
        example_funcs.is_even_advanced(4.5)
    assert example_funcs.is_even_advanced(4)
    assert not example_funcs.is_even_advanced(5)
```

```
$ pytest .
======== test session starts ========
collected 2 items

test/test_even.py ..                [100%]

========= 2 passed in 0.02s =========
```

A second test ran and succeeded

UNIT TESTS

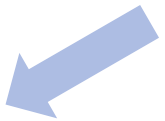
Failing tests

test/test_even.py

…

```
def test_failing():
    assert example_funcs.is_even(5)
```

```
pytest .
========================== test session starts ==========================
collected 3 items

test/test_even.py ..F                                              [100%]
================================== FAILURES ===================================
```



The third test failed

UNIT TESTS

Failing tests

```
pytest .
========================== test session starts ==========================
collected 3 items

test/test_even.py ..F                                              [100%]
================================ FAILURES ================================
_____________________________ test_failing ______________________________
    def test_failing():
>       assert example_funcs.is_even(5)
E       assert False
E        +  where False = <function is_even at 0x101bac820>(5)
E        +    where <function is_even at 0x101bac820> = example_funcs.is_even

test/test_even.py:25: AssertionError
====================== short test summary info ======================
FAILED test/test_even.py::test_failing - assert False
==================== 1 failed, 2 passed in 0.03s ====================
```

UNIT TESTS

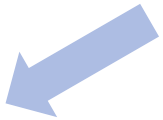
Skipping tests

test/test_even.py

…

```python
@pytest.mark.skip()
def test_failing():
    assert example_funcs.is_even(5)
```

```
pytest .
========================== test session starts ==========================
collected 3 items

test/test_even.py ..s                                              [100%]
==================== 2 passed, 1 skipped in 0.01s =====================
```



The third test is skipped

FURTHER TESTING

- Integration tests:
  - Test interaction of multiple functions; multiple units in general
- System tests & End-To-End tests:
  - Test a whole system / the whole software

Those tests normally occur once the unit tests are finished successfully

Outside of the scope of this talk

OVERVIEW

- Linting & Formatting
- How to do unit-tests with pytest
- Test coverage
- Property based testing
- Conclusion & Further reading

TEST COVERAGE

- How much of the code is tested?
- All untested code is potentially wrong
- We can use `coverage` for that
- Install `pytest-cov` (or add to environment-file, when working with conda/mamba)

```
pytest .
========================= test session starts ==========================
collected 3 items

test/test_even.py ..s                                             [100%]

=================== 2 passed, 1 skipped in 0.01s ====================
```

TEST COVERAGE

Specify directory to calculate coverage

```
pytest . --cov=HiRSE_SoT
========================= test session starts ==========================
collected 3 items

test/test_even.py ..s                                              [100%]

--------- coverage: platform darwin, python 3.10.12-final-0 ----------
Name                            Stmts   Miss  Cover
---------------------------------------------------
HiRSE_SoT/__init__.py               0      0   100%
HiRSE_SoT/example_funcs.py          9      0   100%
HiRSE_SoT/linting.py                3      3     0%
---------------------------------------------------
TOTAL                              12      3    75%


================== 2 passed, 1 skipped in 0.03s ====================
```

Coverage per file

Overall coverage

TEST COVERAGE

Get info which lines are untested

```
pytest . --cov=HiRSE_SoT --cov-report=term-missing
======================== test session starts ========================
collected 3 items

test/test_even.py ..s                                              [100%]

--------- coverage: platform darwin, python 3.10.12-final-0 ----------
Name                          Stmts   Miss  Cover   Missing
-----------------------------------------------------------
HiRSE_SoT/__init__.py             0      0   100%
HiRSE_SoT/example_funcs.py        9      0   100%
HiRSE_SoT/linting.py              3      3     0%   9-11
-----------------------------------------------------------
TOTAL                            12      3    75%


================== 2 passed, 1 skipped in 0.06s ====================
```

Missed lines

OVERVIEW

- Linting & Formatting
- How to do unit-tests with pytest
- Test coverage
- Property based testing
- Conclusion & Further reading

PROPERTY BASED TESTING

- We specified exact input data and output-data (`assert y == function(x)`)
- Risk of missing edge-cases or very repetitive work
- We can use `hypothesis` for that
- Install `hypothesis` (or add to environment-file, when working with conda/mamba)

HiRSE_SoT/random_increase.py

```python
import random


def increase_number(number):
    random_number = random.randint(1, 10)
    return number + random_number
```

test/test_random_increase.py

```python
import hypothesis.strategies as st
from hypothesis import given, example
from HiRSE_SoT import random_increase


@given(st.integers())
@example(5)
def test_random_increase(number):
    result = random_increase.increase_number(number)
    assert isinstance(result, int)
    assert result > number
```

Input-Data

Function that generates arbitrary integers

Specific example

Test now requires a variable

PROPERTY BASED TESTING

```
pytest .
======================== test session starts ========================
collected 4 items

test/test_even.py ..s                                          [ 75%]
test/test_random_increase.py .                                 [100%]

================== 3 passed, 1 skipped in 0.19s ==================
```

Another file
with a single test

PROPERTY BASED TESTING

Previously only tested with integers; now also test floats

test/test_random_increase.py

```
…

@given(st.one_of([st.integers(), st.floats()]))
@example(5)
@example(4.5)
def test_random_increase_float(number):
    result = random_increase.increase_number(number)
    assert isinstance(result, (int, float))
    assert result > number
```

Use one of the following

Function that generates arbitrary floats

Another specific example

Any of the two types is accepted

PROPERTY BASED TESTING

```
pytest .
====================== test session starts ======================
collected 5 items

test/test_even.py ..s                                          [ 60%]
test/test_random_increase.py .F                                [100%]

============================ FAILURES ============================

number = 7.205759403792794e+16

    def test_random_increase_float(number):
        result = random_increase.increase_number(number)
        assert isinstance(result, (int, float))
>       assert result > number
E       assert 7.205759403792794e+16 > 7.205759403792794e+16
E       Falsifying example: test_random_increase_float(number=7.205759403792794e+16)

test/test_random_increase.py:24: AssertionError
==================== short test summary info ====================
FAILED test/test_random_increase.py::test_random_increase_float - assert 7.205759403792794e+16 >
7.205759403792794e+16
============= 1 failed, 3 passed, 1 skipped in 0.20s =============
```

Very large floats do not compare larger

PROPERTY BASED TESTING

test/test_random_increase.py

…

Specify an upper and lower limit

```
@given(st.one_of([st.integers(), st.floats(min_value=-1e15, max_value=1e15)]))
@example(5)
@example(4.5)
def test_random_increase_float(number):
    result = random_increase.increase_number(number)
    assert isinstance(result, (int, float))
    assert result > number
```

```
pytest .
====================== test session starts ======================
platform darwin -- Python 3.10.12, pytest-7.4.0, pluggy-1.2.0
rootdir: /Users/jakob/Code/testing_with_python
plugins: hypothesis-6.82.0, cov-4.1.0
collected 5 items

test/test_even.py ..s                                          [ 60%]
test/test_random_increase.py ..                                [100%]

================== 4 passed, 1 skipped in 0.23s ==================
```

Now, both tests succeed

PROPERTY BASED TESTING

Conclusion

- Used to check properties of data, rather than specific examples
- Great for refactoring code,
  as two function should return the same result for many/all input variables
- Many different types of data can be created, even composite ones
- An addition rather than replacement for testing specific inputs and outputs
- Add `.hypothesis/*` to `.gitignore` as examples are stored there

CONCLUSION

- Linting & Formatting
- How to do unit-tests with pytest
- Test coverage
- Property based testing
- Conclusion & Further reading

CONCLUSION

- Linting & Formatting
  - Find syntactic errors and bad style
  - Helps to (automatically) fix those
  - Easy to add, not specific to task of the code
  - Examples: black, ruff, pylint, flake8
- Unit tests with pytest
  - Finds wrong acting of code (sematic errors)
  - Need to be written specifically for code (no one-fits-all solution)
  - Examples: pytest
- Integration tests, End-To-End tests
  - Check if functions work together as expected
- Test coverage
  - See where untested code is
  - Helps to check all code
- Property-based testing
  - Abstract concept of not testing for specific configuration and values
  - Helps to find edge-cases
  - An addition, rather than replacement of “regular” golden-record unit tests

FURTHER READING

- Ruff:
  - https://astral.sh/ruff
- Flake8:
  - https://flake8.pycqa.org/en/latest/
- Black:
  - https://black.readthedocs.io/en/stable/
- Pytest:
  - https://docs.pytest.org/en/latest/getting-started.html
  - https://docs.pytest.org/en/latest/explanation/goodpractices.html
- Coverage:
  - https://coverage.readthedocs.io/
- Hypothesis:
  - https://hypothesis.readthedocs.io/en/latest/index.html
- Shown Code:
  - https://jugit.fz-juelich.de/rg-rse/testing-with-python

THANK YOU FOR YOUR ATTENTION