© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

pytest tips and tricks for a better testsuite

Florian Bruhin

July 18th, 2023

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

About me

Florian Bruhin <florian@bruhin.software>, The-Compiler, https://bruhin.software

2006 Started programming (QBasic, bash)

2009 – 2013 Apprenticeship in electrical engineering

2011 Started using Python

2013 Started developing qutebrowser, writing tests

2015 Switched to pytest, ended up as a maintainer, started giving courses in companies / at conferences

2016 – 2019 BSc. in Computer Science (OST Rapperswil) Eastern Switzerland University of Applied Sciences

2020 Small one-man company: Bruhin Software

40% employed (OST): Teaching Automation with Python to first-semester students
60% open-source and freelancing (Bruhin Software): Python, pytest, Qt

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

About me

Florian Bruhin <florian@bruhin.software>, The-Compiler, https://bruhin.software

2006 Started programming (QBasic, bash)

2009 – 2013 Apprenticeship in electrical engineering

2011 Started using Python

2013 Started developing qutebrowser, writing tests

2015 Switched to pytest, ended up as a maintainer,
started giving courses in companies / at conferences

2016 – 2019 BSc. in Computer Science (OST Rapperswil)
Eastern Switzerland University of Applied Sciences

2020 Small one-man company: Bruhin Software

40% employed (OST): Teaching Automation with Python to first-semester students
60% open-source and freelancing (Bruhin Software): Python, pytest, Qt

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

About me

Florian Bruhin <florian@bruhin.software>, The-Compiler, https://bruhin.software

2006 Started programming (QBasic, bash)

2009 – 2013 Apprenticeship in electrical engineering

2011 Started using Python

2013 Started developing qutebrowser, writing tests

2015 Switched to pytest, ended up as a maintainer,
started giving courses in companies / at conferences

2016 – 2019 BSc. in Computer Science (OST Rapperswil)
Eastern Switzerland University of Applied Sciences

2020 Small one-man company: Bruhin Software

40% employed (OST): Teaching Automation with Python to first-semester students
60% open-source and freelancing (Bruhin Software): Python, pytest, Qt

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

About me

Florian Bruhin <florian@bruhin.software>, The-Compiler, https://bruhin.software

2006 Started programming (QBasic, bash)

2009 – 2013 Apprenticeship in electrical engineering

2011 Started using Python

2013 Started developing qutebrowser, writing tests

2015 Switched to pytest, ended up as a maintainer,
started giving courses in companies / at conferences

2016 – 2019 BSc. in Computer Science (OST Rapperswil)
Eastern Switzerland University of Applied Sciences

2020 Small one-man company: Bruhin Software

40% employed (OST): Teaching Automation with Python to first-semester students
60% open-source and freelancing (Bruhin Software): Python, pytest, Qt

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Timeline

- 13:45 – 15:15
- 15:30 – 17:00

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Course content

- **About pytest**: Popularity, history overview
- **Parametrization**: Running tests against sets of input/output data
- **Fixtures**: Providing test data, setting up objects, modularity
- **Built-in fixtures**: Temporary files, patching, capturing output, test information
- **Fixtures advanced**: Caching, cleanup/teardown, implicit fixtures, parametrizing
- **Mocking**: Dealing with dependencies which are in our way, monkeypatch and unittest.mock, mocking libraries and helpers, alternatives
- **Plugin tour**: Coverage, distributed testing, output improvements, alternative test syntax, testing C libraries, plugin overview
- **Property-based testing**: Using *hypothesis* to generate test data
- **Writing plugins**: Extending pytest via custom hooks, domain-specific languages

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

pytest

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Setup

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

You...

- **Set up** pytest?

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

You...

- **Set up** pytest?
- Used **virtualenv**?

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

You...

- **Set up** pytest?
- Used **virtualenv**?
- Cloned the repo with example code?

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

You...

- **Set up** pytest?
- Used **virtualenv**?
- Cloned the repo with example code?
- Know what an RPN calculator is?

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Setup overview

- We'll use Python 3.8 or newer, with pytest 7.4 ($\geq$ 7.0 is okay).
  - Use `python3 --version` or `py -3 --version` (Windows) to check your version.
- You can use whatever editor/IDE you'd like – if you don't use one yet, PyCharm (Community Edition) or VS Code are good choices.
- However, we'll first start exploring pytest on the command line, in order to see how it works "under the hood" and explore various commandline arguments.
- Download example code for exercises: https://github.com/The-Compiler/pytest-ep2023

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Setup with PyCharm

- Open `code/` folder as PyCharm project
- Install requirements when prompted
- Open PyCharm terminal at the bottom
- You should be able to run `pytest --version`

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Virtual environments: Isolation of package installs

Virtual environments:

- Provide isolated environments for Python package installs
- Isolate different app/package-install configurations
- Are built into Python since 3.4 (but a separate `virtualenv` tool also exists)

With a virtual environment, we can avoid running `sudo pip install` ... which can mess up your system (on Linux/macOS).

Chris Warrick (`chriswarrick.com`):
"Python Virtual Environments in Five Minutes"

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Using virtual environments

Installing and creating

Install venv:

(Debian-based Linux distributions only, shipped with Python elsewhere)

```
apt install python3-venv
```

Create a local environment (once, can be reused):

```
py -m venv venv
```

```
python3 -m venv .venv
```

(or `virtualenv` instead of `venv`)

This will create a local Python installation in a `venv` or `.venv` folder.
Any dependencies installed with its `pip` will only be available in this environment/folder.

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Using virtual environments

Running commands and activating

Run commands "inside" the environment:

```
venv\Scripts\pip
venv\Scripts\python
venv\Scripts\pytest
```

```
.venv/bin/pip
.venv/bin/python
.venv/bin/pytest
```

Alternatively, **activate** the environment:

(changes `PATH` temporarily, so that `pip`, `python`, `pytest` etc. use the binaries from the virtualenv)

```
venv\Scripts\activate.bat
```

```
Set-ExecutionPolicy Unrestricted -Scope Process
venv\Scripts\Activate
```

```
source .venv/bin/activate
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Installing pytest

Install pytest and other dependencies within the activated environment:

```
pip install -r code/requirements.txt
```

(or just `pip install pytest`, which covers most but not all of the training)

Now let's see if it works:

```
pytest --version
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

The basics

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

You...

- Used **pytest.approx**?

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

You...

- Used **pytest.approx**?
- Know what 0.1 + 0.1 + 0.1 equals (according to ~~Python~~ computers)?

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

You...

- Used **pytest.approx**?
- Know what 0.1 + 0.1 + 0.1 equals (according to ~~Python~~ computers)?
- Used **pytest.raises**?

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

You...

- Used **pytest.approx**?
- Know what 0.1 + 0.1 + 0.1 equals (according to ~~Python~~ computers)?
- Used **pytest.raises**?
- ...with `match=`?

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

You...

- Used **pytest.approx**?
- Know what 0.1 + 0.1 + 0.1 equals (according to ~~Python~~ computers)?
- Used **pytest.raises**?
- ...with `match=`?
- Think pytest is around five years old? Ten? Fifteen? Twenty?

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

You...

- Used **pytest.approx**?
- Know what 0.1 + 0.1 + 0.1 equals (according to ~~Python~~ computers)?
- Used **pytest.raises**?
- ...with `match=`?
- Think pytest is around five years old? Ten? Fifteen? Twenty?

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

You...

- Used **pytest.approx**?
- Know what 0.1 + 0.1 + 0.1 equals (according to ~~Python~~ computers)?
- Used **pytest.raises**?
- ...with `match=`?
- Think pytest is around five years old? Ten? Fifteen? Twenty?

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

You...

- Used **pytest.approx**?
- Know what 0.1 + 0.1 + 0.1 equals (according to ~~Python~~ computers)?
- Used **pytest.raises**?
- ...with `match=`?
- Think pytest is around five years old? Ten? Fifteen? Twenty?

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Some quick history

late 2002	PyPy (alternative Python implementation) was born
mid 2004	New `utest` test framework in PyPy, plain assertions
June 2004	`std` library ("complementary stdlib"): `std.utest`
Sep./Oct. 2004	`std` renamed to `py`, test framework is now `py.test`

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Some quick history

late 2002	PyPy (alternative Python implementation) was born
mid 2004	New `utest` test framework in PyPy, plain assertions
June 2004	`std` library ("complementary stdlib"): `std.utest`
Sep./Oct. 2004	`std` renamed to `py`, test framework is now `py.test`
August 2009	py 1.0.0: plugins, fixtures (funcargs), etc.
November 2010	pytest 2.0.0, released independently from `py`.

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Some quick history

late 2002	PyPy (alternative Python implementation) was born
mid 2004	New `utest` test framework in PyPy, plain assertions
June 2004	`std` library ("complementary stdlib"): `std.utest`
Sep./Oct. 2004	`std` renamed to `py`, test framework is now `py.test`
August 2009	py 1.0.0: plugins, fixtures (funcargs), etc.
November 2010	pytest 2.0.0, released independently from `py`.
August 2016	pytest 3.0.0, new `pytest` entry point (instead of `py.test`)
October 2022	pytest 7.2.0, dropping `py` dependency, vendoring only remaining part (`py.path`)

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Asserting expected exceptions

```python
# rpncalc/utils.py

def calc(a, b, op):
    ...
    elif op == '/':
        return a / b
    raise ValueError("Invalid operator")
```

```python
# basic/test_raises.py

def test_zero_division():
    with pytest.raises(ZeroDivisionError):
        calc(3, 0, '/')
```

Exercise:

- In `basic/test_raises.py`, write another test with `pytest.raises`, to ensure that `ValueError` is raised when calling `calc` with an invalid operator
- Try its `match` argument to check the exception message:
  `with pytest.raises(ValueError, match=r"regex-pattern"):`

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Asserting expected exceptions

```python
# rpncalc/utils.py

def calc(a, b, op):
    ...
    elif op == '/':
        return a / b
    raise ValueError("Invalid operator")
```

```python
# basic/test_raises.py

def test_zero_division():
    with pytest.raises(ZeroDivisionError):
        calc(3, 0, '/')
```

There is also `pytest.warns` to check for warnings:

```python
def test_deprecated_api():
    with pytest.warns(DeprecationWarning, match=[...]):
        [...]
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Comparing floating point numbers

```
def test_add():
    res = calc(0.2, 0.1, '+')
    assert res == 0.3
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Comparing floating point numbers

```
def test_add():
    res = calc(0.2, 0.1, '+')
    assert res == 0.3

E   assert 0.30000000000000004 == 0.3
```

- Floating point numbers have limited precision, thus comparisons via `==` are tricky
- This is a problem in almost every language: `0.30000000000000004.com` (yes, really!)

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Comparing floating point numbers

```python
def test_add():
    res = calc(0.2, 0.1, '+')
    assert res == 0.3
```

```
E   assert 0.30000000000000004 == 0.3
```

```python
# basic/test_approx.py
def test_add():
    res = calc(0.2, 0.1, '+')
    assert res == pytest.approx(0.3)
```

- Floating point numbers have limited precision, thus comparisons via == are tricky
- This is a problem in almost every language: `0.30000000000000004.com` (yes, really!)

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Comparing floating point numbers

```
def test_add():
    res = calc(0.2, 0.1, '+')
    assert res == 0.3
```

```
E   assert 0.30000000000000004 == 0.3
```

```
# basic/test_approx.py
def test_add():
    res = calc(0.2, 0.1, '+')
    assert res == pytest.approx(0.3)
```

- Floating point numbers have limited precision, thus comparisons via == are tricky
- This is a problem in almost every language: `0.30000000000000004.com` (yes, really!)
- `pytest.approx` instead of Python's `math.isclose` gives you nicer output
- Can override tolerance (rel/abs), e.g. 20 ± 2°:
  `assert temperature == pytest.approx(20, abs=2)`

Supports various data types:

- Sequences of numbers (e.g. lists or tuples)
- Dictionary values
- `numpy` arrays

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Marks

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

You...

- Used **pytest.mark.skip**?

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

You...

- Used **pytest.mark.skip**?
- Used **pytest.mark.xfail**?

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

You...

- Used **pytest.mark.skip**?
- Used **pytest.mark.xfail**?
- Used the **parametrize** mark?

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

You...

- Used **pytest.mark.skip**?
- Used **pytest.mark.xfail**?
- Used the **parametrize** mark?
- Used **pytest.param**?

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

You...

- Used **pytest.mark.skip**?
- Used **pytest.mark.xfail**?
- Used the **parametrize** mark?
- Used **pytest.param**?
- Customized **test ids**?

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

You...

- Used **pytest.mark.skip**?
- Used **pytest.mark.xfail**?
- Used the **parametrize** mark?
- Used **pytest.param**?
- Customized **test ids**?
- Used **indirect parametrization**?

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

You...

- Used **pytest.mark.skip**?
- Used **pytest.mark.xfail**?
- Used the **parametrize** mark?
- Used **pytest.param**?
- Customized **test ids**?
- Used **indirect parametrization**?
- Used the **pytestmark** global variable?

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

pytest.mark: Custom marking

Mark functions or classes:

```python
# marking/test_marking.py

@pytest.mark.slow
@pytest.mark.webtest
def test_slow_api():
    time.sleep(1)

@pytest.mark.webtest
def test_api():
    pass

def test_fast():
    pass
```

slow webtest

test_slow_api

webtest

test_api

test_fast

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Marking an entire class

Decorators can be applied to classes as well as functions:

```
@pytest.mark.basic
class TestCalc:

    def test_add(self):
        assert calc(1, 3, '+') == 4

    def test_subtract(self):
        [...]
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Marking an entire test file

To apply a mark to an entire test file, a special `pytestmark` global variable can be set to a mark, or a list of marks.

```
pytestmark = pytest.mark.skipif(
    sys.platform == "win32",
    reason="Linux only"
)
```

The same technique can be used to skip all tests in a file unconditionally:

```
pytestmark = pytest.mark.skip("Work in progress")
```

If all tests in a file should be skipped if a library was not found, there's a helper. This is the same as `import pexpect`, but the module is skipped on `ImportError`:

```
pexpect = pytest.importorskip("pexpect")
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Parametrizing tests

Tests can be parametrized to run them with various values:

```
# marking/test_parametrization.py

@pytest.mark.parametrize("a, b, expected", [
    (1, 1, 3),
    (1, 2, 3),
    (2, 3, 5),
])
def test_add(a, b, expected):
    assert calc(a, b, '+') == expected

@pytest.mark.parametrize(
    "op", ['+', '-', '*', '/', '**'])
def test_smoke(op):
    calc(1, 2, op)
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Marking single parameters

If we want to e.g. xfail a single parameter combination in a parametrized test, the mark can be attached to a single value via `pytest.param`:

```
@pytest.mark.parametrize(
    "a, b, op, expected", [
        pytest.param(2, 5, '**', 32),
        (1, 2, '+', 3),
        (3, 1, '-', 2),
])
def test_calc(a, b, op, expected):
    assert calc(a, b, op) == expected
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Marking single parameters

If we want to e.g. xfail a single parameter combination in a parametrized test, the mark can be attached to a single value via `pytest.param`:

```
# marking/test_parametrization_marks.py

@pytest.mark.parametrize(
    "a, b, op, expected", [
    pytest.param(
        2, 5, '**', 32,
        marks=pytest.mark.xfail(reason=...),
    ),
    (1, 2, '+', 3),
    (3, 1, '-', 5),
])
def test_calc(a, b, op, expected):
    assert calc(a, b, op) == expected
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Changing test IDs

We can use `pytest.param(..., id="...")` to override the auto-generated test ID:

```
@pytest.mark.parametrize("a, b, op, expected", [
    pytest.param(1, 2, '+', 3),
    pytest.param(3, 1, '-', 5),
])
def test_calc(a, b, op, expected):
    assert calc(a, b, op) == expected
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Changing test IDs

We can use `pytest.param(..., id="...")` to override the auto-generated test ID:

```
@pytest.mark.parametrize("a, b, op, expected", [
    pytest.param(1, 2, '+', 3, id="add"),
    pytest.param(3, 1, '-', 5, id="sub"),
])
def test_calc(a, b, op, expected):
    assert calc(a, b, op) == expected
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Changing test IDs

We can use `pytest.param(..., id="...")` to override the auto-generated test ID:

```
@pytest.mark.parametrize("a, b, op, expected", [
    pytest.param(1, 2, '+', 3, id="add"),
    pytest.param(3, 1, '-', 5, id="sub"),
])
def test_calc(a, b, op, expected):
    assert calc(a, b, op) == expected
```

Or pass a list as `ids=` keyword argument to `@pytest.mark.parametrize`:

```
@pytest.mark.parametrize("a, b, op, expected", [
    (1, 2, '+', 3), (3, 1, '-', 5),
], ids=["add", "sub"])
def test_calc(a, b, op, expected):
    assert calc(a, b, op) == expected
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Changing test IDs

We can use `pytest.param(..., id="...")` to override the auto-generated test ID:

```
@pytest.mark.parametrize("a, b, op, expected", [
    pytest.param(1, 2, '+', 3, id="add"),
    pytest.param(3, 1, '-', 5, id="sub"),
])
def test_calc(a, b, op, expected):
    assert calc(a, b, op) == expected
```

Or pass a list as `ids=` keyword argument to `@pytest.mark.parametrize`:

```
@pytest.mark.parametrize("a, b, op, expected", [
    (1, 2, '+', 3), (3, 1, '-', 5),
], ids=["add", "sub"])
def test_calc(a, b, op, expected):
    assert calc(a, b, op) == expected
```

We can also pass a callable to generate IDs.

e.g. for a list:
`ids=", ".join`

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Trick: Using dataclasses with parametrize

```
@dataclass
class CalcCase:
    name: str
    a: int
    b: int
    result: int
    op: str = "+"
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Trick: Using dataclasses with parametrize

```python
@dataclass
class CalcCase:
    name: str
    a: int
    b: int
    result: int
    op: str = "+"
```

```python
@pytest.mark.parametrize("tc", [
    CalcCase("add", a=1, b=2, result=3),
    CalcCase("add-neg", a=-2, b=-3, result=-5),
    CalcCase("sub", a=2, b=1, op="-", result=1),
], ids=lambda tc: tc.name)
def test_calc(tc):
    assert calc(tc.a, tc.b, tc.op) == tc.result
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Trick: Using dataclasses with parametrize

```python
@dataclass
class CalcCase:
    name: str
    a: int
    b: int
    result: int
    op: str = "+"
```

```python
@pytest.mark.parametrize("tc", [
    CalcCase("add", a=1, b=2, result=3),
    CalcCase("add-neg", a=-2, b=-3, result=-5),
    CalcCase("sub", a=2, b=1, op="-", result=1),
], ids=lambda tc: tc.name)
def test_calc(tc):
    assert calc(tc.a, tc.b, tc.op) == tc.result
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Trick: Using dataclasses with parametrize

```python
@dataclass
class CalcCase:
    name: str
    a: int
    b: int
    result: int
    op: str = "+"
```

```python
@pytest.mark.parametrize("tc", [
    CalcCase("add", a=1, b=2, result=3),
    CalcCase("add-neg", a=-2, b=-3, result=-5),
    CalcCase("sub", a=2, b=1, op="-", result=1),
], ids=lambda tc: tc.name)
def test_calc(tc):
    assert calc(tc.a, tc.b, tc.op) == tc.result
```

- Easy setting of test names – could even write a custom `def __str__(self):` and use `ids=str`
- Thanks to default arguments, we only need to specify values that differ from the default (no `op="+"`)
- Better type safety, better autocompletion
- More readability for complex test cases

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Permutations with parametrize

If we stack `@pytest.mark.parametrize` multiple times, we get all permutations:

```
@pytest.mark.parametrize("a", [1, 2])
@pytest.mark.parametrize("b", [3, 4])
def test_permutations(a, b):
    assert calc(a, b, '+') == a + b
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Skipping or "xfailing" tests

Skip a test if:

- It cannot run at all on a certain platform
- It cannot run because a dependency is missing

⇒ Test function is not run, result is "skipped" (`s`)

Use `@pytest.mark.skip` (instead of `skipif`) for unconditional skipping.

```
@pytest.mark.skipif(
    # condition
    sys.platform == "win32",
    # text shown with -v
    reason="Linux only",
)
def test_linux():
    [...]
```

"xfail" ("expected to fail") a test if:

- The implementation is currently lacking
- It fails on a certain platform but should work

⇒ Test function is run, but result is "xfailed" (`x`), instead of failed (`F`). Unexpected pass: XPASS (`X`).

```
@pytest.mark.xfail(
    # condition optional
    reason="see #1234",
)
def test_new_api():
    [...]
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

The strict option for xfail

If a test marked `xfail` ("expected to fail") passes, the result is an `XPASS` (counts as passed test) by default. This can by changed by using:

```
@pytest.mark.xfail(reason="...", strict=True)
```

or globally by setting:

```
[pytest]
xfail_strict=true
```

Then, a test which is expected to fail but passes results in a failing test.
This can be useful to let pytest alert you about accidentally fixed bugs!

Note: This won't work with imperative `pytest.xfail(...)`, as test gets skipped.

If you're dealing with flaky tests (sometimes passing, sometimes failing),
it's better to use a plugin such as `pytest-rerunfailures` instead.

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Expanding the calculator example

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

History

- Using a calculator without needing a = key, and without parentheses
- Makes it much easier to implement, using a stack data structure
- Based on the Polish notation, invented by Jan Łukasiewicz in 1924
- Used by all HP calculators in the 1970s–80s, still used by some today
- Displayed here: HP 12C financial calculator, introduced in 1981, still in production today (HPs longest and best-selling product)

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

Explanation

$$1 + 2$$

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

Explanation

$$5 \cdot (1 + 2)$$

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

In Python

In `code/`, using `python -m rpncalc.rpn_v1`:

```
> 1
> 2
> +
3
> 5
> *
15
> q
```

Exercise: Run the calculator, play around a bit, and try to break it.
Take a first look at its code (`rpncalc/rpn_v1.py`), it'll be explained on the next slide.

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

Code

```python
# rpncalc/rpn_v1.py

from rpncalc.utils import calc

class RPNCalculator:

    def __init__(self):
        self.stack = []

    def run(self):
        while True:
            inp = input('> ')
            if inp == 'q':
                return
            self.evaluate(inp)
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

Code

```python
# rpncalc/rpn_v1.py

from rpncalc.utils import calc

class RPNCalculator:

    def __init__(self):
        self.stack = []

    def run(self):
        while True:
            inp = input('> ')
            if inp == 'q':
                return
            self.evaluate(inp)
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

Code

```python
# rpncalc/rpn_v1.py

from rpncalc.utils import calc

class RPNCalculator:

    def __init__(self):
        self.stack = []

    def run(self):
        while True:
            inp = input('> ')
            if inp == 'q':
                return
            self.evaluate(inp)
```

```python
    def evaluate(self, inp):
        if inp.isdigit():
            n = float(inp)
            self.stack.append(n)
        elif inp in '+-*/':
            b = self.stack.pop()
            a = self.stack.pop()
            res = calc(a, b, inp)
            self.stack.append(res)
            print(res)
```

```python
if __name__ == "__main__":
    rpn = RPNCalculator()
    rpn.run()
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

Code

```python
# rpncalc/rpn_v1.py

from rpncalc.utils import calc

class RPNCalculator:

    def __init__(self):
        self.stack = []

    def run(self):
        while True:
            inp = input('> ')
            if inp == 'q':
                return
            self.evaluate(inp)
```

```python
    def evaluate(self, inp):
        if inp.isdigit():
            n = float(inp)
            self.stack.append(n)
        elif inp in '+-*/':
            b = self.stack.pop()
            a = self.stack.pop()
            res = calc(a, b, inp)
            self.stack.append(res)
            print(res)

if __name__ == "__main__":
    rpn = RPNCalculator()
    rpn.run()
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

Code

```python
# rpncalc/rpn_v1.py

from rpncalc.utils import calc

class RPNCalculator:

    def __init__(self):
        self.stack = []

    def run(self):
        while True:
            inp = input('> ')
            if inp == 'q':
                return
            self.evaluate(inp)

    def evaluate(self, inp):
        if inp.isdigit():
            n = float(inp)
            self.stack.append(n)
        elif inp in '+-*/':
            b = self.stack.pop()
            a = self.stack.pop()
            res = calc(a, b, inp)
            self.stack.append(res)
            print(res)

if __name__ == "__main__":
    rpn = RPNCalculator()
    rpn.run()
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

Code

```python
# rpncalc/rpn_v1.py

from rpncalc.utils import calc

class RPNCalculator:

    def __init__(self):
        self.stack = []

    def run(self):
        while True:
            inp = input('> ')
            if inp == 'q':
                return
            self.evaluate(inp)

    def evaluate(self, inp):
        if inp.isdigit():
            n = float(inp)
            self.stack.append(n)
        elif inp in '+-*/':
            b = self.stack.pop()
            a = self.stack.pop()
            res = calc(a, b, inp)
            self.stack.append(res)
            print(res)

if __name__ == "__main__":
    rpn = RPNCalculator()
    rpn.run()
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

Finding out what's broken

Let's focus on the `evaluate` method:

```
class RPNCalculator:
    ...
    def evaluate(self, inp):
        if inp.isdigit():
            n = float(inp)
            self.stack.append(n)
        elif inp in '+-*/':
            b = self.stack.pop()
            a = self.stack.pop()
            res = calc(a, b, inp)
            self.stack.append(res)
```

Spot the bugs!

1. Which inputs currently cause unhandled exceptions?
2. Which inputs cause confusing behavior?
3. What should probably work, but doesn't?

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

Towards an improved version

- Fix bugs and add additional error handling:
  - Allow negative numbers and floating-point inputs, not just `.isdigit()`
  - Fix `+-` being treated as valid input due to `elif inp in "+-*/":`
  - Print error when using an invalid operator
  - Handle `ZeroDivisionError` when dividing by zero, and `IndexError` with < 2 elements on stack
- Allow passing a `Config` object to the calculator, with a custom prompt, instead of `"> "`
- Support multiple inputs on one line, e.g. `2 1 +`

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

Code: Fixing bugs, improving error handling

```python
# rpncalc/rpn_v1.py
...

def evaluate(self, inp):
    if inp.isdigit():
        n = float(inp)
        self.stack.append(n)
    elif inp in '+-*/':
        ...
```

```python
# rpncalc/rpn_v2.py
...

def evaluate(self, inp):
    try:
        self.stack.append(float(inp))
        return
    except ValueError:
        pass

    if inp not in ['+', '-', '*', '/']:
        ...
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

Code: Fixing bugs, improving error handling

```python
# rpncalc/rpn_v1.py
...

def evaluate(self, inp):
    if inp.isdigit():
        n = float(inp)
        self.stack.append(n)
    elif inp in '+-*/':
        ...
```

```python
# rpncalc/rpn_v2.py
...

def evaluate(self, inp):
    try:
        self.stack.append(float(inp))
        return
    except ValueError:
        pass

    if inp not in ['+', '-', '*', '/']:
        ...
```

```python
# rpncalc/test_rpn_v2.py

@pytest.mark.parametrize(
    "n", [1.5, -1])
def test_number_input(rpn, n):
    rpn.evaluate(str(n))
    assert rpn.stack == [n]
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

Code: Fixing bugs, improving error handling

```python
# rpncalc/rpn_v1.py
...

def evaluate(self, inp):
    if inp.isdigit():
        n = float(inp)
        self.stack.append(n)
    elif inp in '+-*/':
        ...
```

```python
# rpncalc/rpn_v2.py
...

def evaluate(self, inp):
    try:
        self.stack.append(float(inp))
        return
    except ValueError:
        pass

    if inp not in ['+', '-', '*', '/']:
        ...
```

```python
# rpncalc/test_rpn_v2.py

@pytest.mark.parametrize(
    "n", [1.5, -1])
def test_number_input(rpn, n):
    rpn.evaluate(str(n))
    assert rpn.stack == [n]
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Fixtures

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

You...

- Used pytest **fixtures**?

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

You...

- Used pytest **fixtures**?
- Used **conftest.py**?

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

You...

- Used pytest **fixtures**?
- Used **conftest.py**?
- Used **“yield” in a fixture**?

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

You...

- Used pytest **fixtures**?
- Used **conftest.py**?
- Used **"yield" in a fixture**?
- Used **tmp_path** or **monkeypatch**?

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

You...

- Used pytest **fixtures**?
- Used **conftest.py**?
- Used **"yield" in a fixture**?
- Used **tmp_path** or **monkeypatch**?
- Used **scope=...** or **autouse=** for a fixture?

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

You...

- Used pytest **fixtures**?
- Used **conftest.py**?
- Used **"yield" in a fixture**?
- Used **tmp_path** or **monkeypatch**?
- Used **scope=...** or **autouse=** for a fixture?
- Used **autouse=True for a fixture**?

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

You...

- Used pytest **fixtures**?
- Used **conftest.py**?
- Used **"yield" in a fixture**?
- Used **tmp_path** or **monkeypatch**?
- Used **scope=...** or **autouse=** for a fixture?
- Used **autouse=True for a fixture**?
- **Parametrized a fixture**?

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

Code: Adding a Config object

```python
# rpncalc/rpn_v2.py

from rpncalc.utils import calc, Config

class RPNCalculator:

    def __init__(self, config):
        self.config = config
        self.stack = []

...

if __name__ == "__main__":
    config = Config()
    rpn = RPNCalculator(config)
    rpn.run()
```

```python
# rpncalc/utils.py

class Config:

    def __init__(self, prompt=">"):
        self.prompt = prompt

    ...
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Example of pytest fixture injection

pytest calls the fixture function to inject a dependency into the test function:

```
# fixtures/test_fixture.py
@pytest.fixture
def rpn() -> RPNCalculator:
    return RPNCalculator(Config())

def test_empty_stack(rpn: RPNCalculator):
    assert rpn.stack == []
```

Imagine pytest running your test like:

```
test_empty_stack(rpn=rpn())
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Showing available fixtures

`--fixtures`	Show all defined fixtures with their docstrings.
`--fixtures-per-test`	Show the fixtures used, grouped by test.

Good practice: Write a docstring for your fixtures, consider adding type annotations:

```
@pytest.fixture
def rpn() -> RPNCalculator:
    """A RPN calculator with a default config."""
    [...]

def test_rpn(rpn: RPNCalculator):
    [...]
```

Output:

```
----------------- fixtures defined from test_fixture -----------------
rpn -- fixtures/test_fixture.py:7
    A RPN calculator with a default config.
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Caching fixture results

Fixture functions can declare a caching scope:

```
@pytest.fixture
def rpn() -> RPNCalculator:
    time.sleep(2)
    return RPNCalculator(Config())

def test_a(rpn: RPNCalculator):
    rpn.stack.append(42)
    assert rpn.stack == [42]

def test_b(rpn: RPNCalculator):
    assert not rpn.stack
```

Function scope:

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Caching fixture results

Fixture functions can declare a caching scope:

```
# fixtures/test_fixture_scope.py

@pytest.fixture(scope="function")
def rpn() -> RPNCalculator:
    time.sleep(2)
    return RPNCalculator(Config())

def test_a(rpn: RPNCalculator):
    rpn.stack.append(42)
    assert rpn.stack == [42]

def test_b(rpn: RPNCalculator):
    assert not rpn.stack
```

Function scope:

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Caching fixture results

Fixture functions can declare a caching scope:

```
# fixtures/test_fixture_scope.py

@pytest.fixture(scope="module")
def rpn() -> RPNCalculator:
    time.sleep(2)
    return RPNCalculator(Config())

def test_a(rpn: RPNCalculator):
    rpn.stack.append(42)
    assert rpn.stack == [42]

def test_b(rpn: RPNCalculator):
    assert not rpn.stack
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Caching fixture results

Fixture functions can declare a caching scope:

```
# fixtures/test_fixture_scope.py

@pytest.fixture(scope="module")
def rpn() -> RPNCalculator:
    time.sleep(2)
    return RPNCalculator(Config())

def test_a(rpn: RPNCalculator):
    rpn.stack.append(42)
    assert rpn.stack == [42]

def test_b(rpn: RPNCalculator):
    assert not rpn.stack
```

Beware:

+ Faster tests (4s → 2s)

– Less isolation between tests:

```
[...]::test_a PASSED
[...]::test_b FAILED
```

```
def test_b(rpn: RPNCalculator):
>   assert not rpn.stack
E   assert not [42]
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Caching fixture results

Fixture functions can declare a caching scope:

```python
# fixtures/test_fixture_scope.py

@pytest.fixture(scope="module")
def rpn() -> RPNCalculator:
    time.sleep(2)
    return RPNCalculator(Config())

def test_a(rpn: RPNCalculator):
    rpn.stack.append(42)
    assert rpn.stack == [42]

def test_b(rpn: RPNCalculator):
    assert not rpn.stack
```

Available scopes:

- `"function"` (default)
- `"class"`
- `"module"`
- `"package"`
- `"session"`

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Caching fixture results

Fixture functions can declare a caching scope:

```python
# fixtures/test_fixture_scope.py

@pytest.fixture(scope="module")
def rpn() -> RPNCalculator:
    time.sleep(2)
    return RPNCalculator(Config())

def test_a(rpn: RPNCalculator):
    rpn.stack.append(42)
    assert rpn.stack == [42]

def test_b(rpn: RPNCalculator):
    assert not rpn.stack
```

Available scopes:

- `"function"` (default)
- `"class"`
- `"module"`
- `"package"`
- `"session"`

Can also pass a callable to `scope` to dynamically determine the scope.
It will get called with the fixture name and the `pytest.Config` object.

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Caching fixture results

Combining scopes

```python
# fixtures/test_fixture_scope_reset.py

@pytest.fixture(scope="module")
def rpn_instance() -> RPNCalculator:
    time.sleep(2)
    return RPNCalculator(Config())

@pytest.fixture
def rpn(
    rpn_instance: RPNCalculator,
) -> RPNCalculator:
    rpn_instance.stack.clear()
    return rpn_instance
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Caching fixture results

Combining scopes

```python
# fixtures/test_fixture_scope_reset.py

@pytest.fixture(scope="module")
def rpn_instance() -> RPNCalculator:
    time.sleep(2)
    return RPNCalculator(Config())

@pytest.fixture
def rpn(
    rpn_instance: RPNCalculator,
) -> RPNCalculator:
    rpn_instance.stack.clear()
    return rpn_instance
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Caching fixture results

Combining scopes

```python
# fixtures/test_fixture_scope_reset.py

@pytest.fixture(scope="module")
def rpn_instance() -> RPNCalculator:
    time.sleep(2)
    return RPNCalculator(Config())

@pytest.fixture
def rpn(
    rpn_instance: RPNCalculator,
) -> RPNCalculator:
    rpn_instance.stack.clear()
    return rpn_instance
```

```python
def test_a(rpn: RPNCalculator):
    rpn.stack.append(42)
    assert rpn.stack == [42]

def test_b(rpn: RPNCalculator):
    assert not rpn.stack
```

```
[...]::test_a PASSED
[...]::test_b PASSED

===== 2 passed in 2.00s =====
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Doing cleanup with yield

A fixture can use `yield` instead of `return` to do cleanup:

```python
# fixtures/test_yield_fixture.py

@pytest.fixture
def connected_client() -> Iterator[Client]:
    client = Client()
    client.connect()
    yield client
    client.disconnect()

def test_client(connected_client: Client):
    print("in the test")
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Doing cleanup with yield

A fixture can use `yield` instead of `return` to do cleanup:

```python
# fixtures/test_yield_fixture.py

@pytest.fixture
def connected_client() -> Iterator[Client]:
    client = Client()
    client.connect()
    yield client
    client.disconnect()

def test_client(connected_client: Client):
    print("in the test")
```

Demo:

- Add a couple of tests using `connected_client`
- Observe teardown behaviour using `-s` and/or `--setup-show`
- Modify fixture scope, check how the behavior changes

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Alternative: request.addfinalizer

Fixture functions can use `request.addfinalizer` to register teardown functions:

```
# fixtures/test_fixture_finalizer.py
@pytest.fixture
def connected_client(request: pytest.FixtureRequest) -> Client:
    client = Client()
    client.connect()
    request.addfinalizer(client.disconnect)
    return client
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Builtin fixtures

pytest provides builtin fixtures:

`tmp_path` / `tmpdir`	A fresh empty directory for each test invocation
`monkeypatch`	Temporarily modify state for test duration
`capsys` and `capfd`	Capturing stdout/stderr in a test
`caplog`	Capturing logging output from a test
`request`	Get information about the current test
...	

More fixtures are provided by plugins.

Use `pytest --fixtures` to see available fixtures with docs

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Builtin fixtures

tmp_path / tmpdir

A **fresh empty directory** for each test invocation

- Store generated input files for tested code
- Store output files, e.g. measurement data, screenshots, etc.
- Access data even after pytest run is done, but no need for manual cleanup

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Builtin fixtures

tmp_path / tmpdir

A **fresh empty directory** for each test invocation

- Store generated input files for tested code
- Store output files, e.g. measurement data, screenshots, etc.
- Access data even after pytest run is done, but no need for manual cleanup

`tmp_path`: Based on Python's `pathlib.Path`, a more object-oriented and convenient `os.path` alternative:

```
out_dir_path = pathlib.Path("output")
out_dir_path.mkdir(exist_ok=True)
out_file_path = out_dir_path / "output.txt"
out_file_path.write_text("Hello world!")
```

`tmpdir`:

Based on `py.path.local` from the `py` library (`pylib`) instead, since `pathlib` only exists since Python 3.4.

Should not be used in new code.

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

Code: Saving and loading config objects

```python
# rpncalc/utils.py
class Config:
    ...

    def load(self, path: pathlib.Path) -> None:
        parser = configparser.ConfigParser()
        parser.read(path)
        self.prompt = parser["rpncalc"]["prompt"]
```

rpncalc.ini

```ini
[rpncalc]
prompt = rpn>
```

Loading

```python
ini_path = Path(
  "rpncalc.ini")
c = Config()
c.load(ini_path)
```

Usage

```
rpn> 1
rpn> 2
rpn> +
3
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

Code: Saving and loading config objects

```python
# rpncalc/utils.py
class Config:
    ...

    def load(self, path: pathlib.Path) -> None:
        parser = configparser.ConfigParser()
        parser.read(path)
        self.prompt = parser["rpncalc"]["prompt"]

    def save(self, path: pathlib.Path) -> None:
        parser = configparser.ConfigParser()
        parser["rpncalc"] = {"prompt": self.prompt}
        with path.open("w") as f:
            parser.write(f)
```

Saving

```python
ini_path = Path(
  "rpncalc.ini")
c = Config()
c.prompt = "calc>"
c.save(ini_path)
```

rpncalc.ini

```ini
[rpncalc]
prompt = calc>
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Builtin fixtures

tmp_path

```python
# fixtures/test_builtin_tmp_path.py

@pytest.fixture
def config_path(tmp_path: Path) -> Path:
    lines = ["[rpncalc]", "prompt = rpn>"]
    path = tmp_path / "calc.ini"
    path.write_text("\n".join(lines))
    return path

def test_config_load(
    config_path: Path, config: Config
):
    print(config_path)
    assert False    # to show output
```

```
/tmp/      (system-wide temp directory)
└── pytest-of-florian/
    ├── pytest-1/
    │   ├── test_config_load0/
    │   │   └── calc.ini
    │   └── test_other0/
    └── pytest-2/
        ├── test_config_load0/
        │   └── calc.ini
        └── test_other0/
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Builtin fixtures

tmp_path

```python
# fixtures/test_builtin_tmp_path.py

@pytest.fixture
def config_path(tmp_path: Path) -> Path:
    lines = ["[rpncalc]", "prompt = rpn>"]
    path = tmp_path / "calc.ini"
    path.write_text("\n".join(lines))
    return path

def test_config_load(
    config_path: Path, config: Config
):
    print(config_path)
    assert False   # to show output
```

Demo:

- Complete `test_config_load`: Use the `config` fixture, load config file, ensure attribute was changed.
- Complete `test_config_save` too, using `tmp_path` as output directory, then reading and asserting on the contents.

pathlib.Path docs:
docs.python.org/3/library/pathlib.html

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Builtin fixtures

monkeypatch

The `monkeypatch` fixture allows to temporarily change state for the **duration of a test function execution**:

- Modify attributes on objects, classes or modules (`setattr`, `delattr`)
  Also works with functions/methods, as those are attributes of the respective module/class/instance too.
- Modify environment variables (`setenv`, `delenv`)
- Modify dictionaries (`setitem`, `delitem`)
- Change current directory (`chdir`)
- Prepend to `sys.path` for importing (`syspath_prepend`)

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

Code: Supporting multiple inputs

```python
# rpncalc/rpn_v1.py

class RPNCalculator:
  ...

  def run(self):
      while True:
          inp = input('> ')
          if inp == 'q':
              return
          self.evaluate(inp)
```

```python
# rpncalc/rpn_v2.py

class RPNCalculator:
    ...

  def get_inputs(self):
      inp = input(self.config.prompt + ' ')
      return inp.split()

  def run(self):
      while True:
          for inp in self.get_inputs():
              if inp == 'q':
                  return
              self.evaluate(inp)
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Builtin fixtures

monkeypatch: How to not do things

```python
def print_info():
    path = os.environ.get("PATH", "")
    print(f"platform: {sys.platform}")
    print(f"PATH: {path}")

def test_one():
    sys.platform = "MonkeyOS"     # don't do this!
    os.environ["PATH"] = "/zoo"   # don't do this!
    print_info()
    assert False

def test_two():
    print_info()
    assert False
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Builtin fixtures

monkeypatch

```python
# fixtures/test_builtin_monkeypatch.py
def print_info():
    path = os.environ.get("PATH", "")
    print(f"platform: {sys.platform}")
    print(f"PATH: {path}")

def test_one(monkeypatch: pytest.MonkeyPatch):
    monkeypatch.setattr(sys, "platform", "MonkeyOS")
    monkeypatch.setenv("PATH", "/zoo")
    print_info()
    assert False

def test_two():
    print_info()
    assert False
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Builtin fixtures

monkeypatch

```
# rpncalc/rpn_v2.py
class RPNCalculator:
    ...

    def get_inputs(self):
        inp = input(self.config.prompt + ' ')
        return inp.split()

    def run(self):
        while True:
            for inp in self.get_inputs():
                if inp == 'q':
                    return
                self.evaluate(inp)
```

```
monkeypatch.setattr(target, "name", value)
```

Exercise:

- Add a new test to `test_rpn_v2.py`, which will test `run`.
- Use the `monkeypatch` fixture to replace the `get_inputs` method.
- Replace it by a function that returns a fixed list of inputs, ending in `"q"`. You could define a `def fake_get_inputs():` or use `lambda: [...]`.
- Leave `rpn_v2.py` as-is!

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Builtin fixtures

Capturing

`capsys` Capturing stdout/stderr in a test by overriding `sys.stdout`
e.g. for `print(...)`

`capfd` Capturing stdout/stderr in a test at file descriptor level
e.g. for subprocesses or libraries printing from C code

`caplog` Capturing logging output from a test

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Builtin fixtures

Capturing

`capsys` Capturing stdout/stderr in a test by overriding `sys.stdout`
e.g. for `print(...)`

`capfd` Capturing stdout/stderr in a test at file descriptor level
e.g. for subprocesses or libraries printing from C code

`caplog` Capturing logging output from a test

Background: Every process has *two* output streams: `stdout` and `stderr`.

```
$ ls /home doesnotexist
ls: cannot access 'doesnotexist': No such file or directory
/home:
username
```

```
$ ls /home doesnotexist > /dev/null
ls: cannot access 'doesnotexist': No such file or directory
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

Code: Fixing bugs, improving error handling

```python
# rpncalc/rpn_v2.py
...

def evaluate(self, inp):
    try:
        self.stack.append(float(inp))
        return
    except ValueError:
        pass

    if inp not in ['+', '-', '*', '/']:
        print(f'Invalid input: {inp}')
        return

    if len(self.stack) < 2:
        print('Not enough operands')
        return

    b = self.stack.pop()
    a = self.stack.pop()

    try:
        res = calc(a, b, inp)
    except ZeroDivisionError:
        print('Division by zero')
        return

    self.stack.append(res)
    print(res)
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

Code: Fixing bugs, improving error handling

```python
# rpncalc/rpn_v2.py
...

def evaluate(self, inp):
    try:
        self.stack.append(float(inp))
        return
    except ValueError:
        pass

    if inp not in ['+', '-', '*', '/']:
        print(f'Invalid input: {inp}')
        return
```

```python
if len(self.stack) < 2:
    print('Not enough operands')
    return

b = self.stack.pop()
a = self.stack.pop()

try:
    res = calc(a, b, inp)
except ZeroDivisionError:
    print('Division by zero')
    return

self.stack.append(res)
print(res)
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

Code: Fixing bugs, improving error handling

```python
# rpncalc/rpn_v2.py
...

def evaluate(self, inp):
    try:
        self.stack.append(float(inp))
        return
    except ValueError:
        pass

    if inp not in ['+', '-', '*', '/']:
        print(f'Invalid input: {inp}')
        return

    if len(self.stack) < 2:
        print('Not enough operands')
        return

    b = self.stack.pop()
    a = self.stack.pop()

    try:
        res = calc(a, b, inp)
    except ZeroDivisionError:
        print('Division by zero')
        return

    self.stack.append(res)
    print(res)
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

Tests: Error handling

```python
# rpncalc/test_rpn_v2.py

@pytest.mark.parametrize("op", ["**", "+-"])
def test_unknown_operator(rpn, op):
    rpn.stack = [1, 2]
    rpn.evaluate(op)  # FIXME how to test that this prints an error?

def test_division_by_zero(rpn):
    rpn.stack = [1, 0]
    rpn.evaluate("/")  # FIXME how to test that this prints an error?

def test_not_enough_operands(rpn):
    rpn.stack = [1]
    rpn.evaluate("+")  # FIXME how to test that this prints an error?
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

Tests: Error handling

```python
# rpncalc/test_rpn_v2.py

@pytest.mark.parametrize("op", ["**", "+-"])
def test_unknown_operator(rpn, op):
    rpn.stack = [1, 2]
    rpn.evaluate(op)  # FIXME how to test that this prints an error?

def test_division_by_zero(rpn):
    rpn.stack = [1, 0]
    rpn.evaluate("/")  # FIXME how to test that this prints an error?

def test_not_enough_operands(rpn):
    rpn.stack = [1]
    rpn.evaluate("+")  # FIXME how to test that this prints an error?
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

Tests: Error handling

```python
# rpncalc/test_rpn_v2.py

@pytest.mark.parametrize("op", ["**", "+-"])
def test_unknown_operator(rpn, op):
    rpn.stack = [1, 2]
    rpn.evaluate(op)  # FIXME how to test that this prints an error?

def test_division_by_zero(rpn):
    rpn.stack = [1, 0]
    rpn.evaluate("/")  # FIXME how to test that this prints an error?

def test_not_enough_operands(rpn):
    rpn.stack = [1]
    rpn.evaluate("+")  # FIXME how to test that this prints an error?
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Builtin fixtures

Capturing

`capsys` overrides `sys.stdout/err`:

```python
# fixtures/test_builtin_capsys.py
def test_output(capsys):
    print("Hello World")
    out, err = capsys.readouterr()
    assert out == "Hello World\n"
```

`capfd` captures at file descriptor level:

```python
# fixtures/test_builtin_capfd.py
def test_output(capfd):
    subprocess.run(["ls"])
    out, err = capfd.readouterr()
    assert out == "..."
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Builtin fixtures

Capturing

`capsys` overrides `sys.stdout/err`:

```
# fixtures/test_builtin_capsys.py
def test_output(capsys):
    print("Hello World")
    out, err = capsys.readouterr()
    assert out == "Hello World\n"
```

`capfd` captures at file descriptor level:

```
# fixtures/test_builtin_capfd.py
def test_output(capfd):
    subprocess.run(["ls"])
    out, err = capfd.readouterr()
    assert out == "..."
```

`caplog` lets you access logging records:

```
# fixtures/test_builtin_caplog.py
def test_output(caplog):
    logging.warning("Something failed")
    assert caplog.messages == ["Something failed"]
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Builtin fixtures

Capturing

`capsys` overrides `sys.stdout/err`:

```
# fixtures/test_builtin_capsys.py
def test_output(capsys):
    print("Hello World")
    out, err = capsys.readouterr()
    assert out == "Hello World\n"
```

`capfd` captures at file descriptor level:

```
# fixtures/test_builtin_capfd.py
def test_output(capfd):
    subprocess.run(["ls"])
    out, err = capfd.readouterr()
    assert out == "..."
```

Exercise:

- Take a look at the revised `evaluate` method in `rpncalc/rpn_v2.py`, with added error handling.
- Pick one of the incomplete tests in `test_rpn_v2.py`.
- Complete it by capturing the printed error message and asserting on it.

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

Loading config based on environment

If `RPNCALC_CONFIG_DIR` is set as an environment variable, calling `config.load_env()` will load the config file from *`RPNCALC_CONFIG_DIR`*`/rpncalc.ini`.

If that file does not exist (but the environment variable is set), `FileNotFoundError` is raised.

Environment variables can be set globally system-wide, or temporarily in a shell/notebook:

Linux	`export RPNCALC_CONFIG_DIR=~/.config/rpncalc`
cmd.exe	`set RPNCALC_CONFIG_DIR=%LOCALAPPDATA%\rpncalc`
Powershell	`$env:RPNCALC_CONFIG_DIR=$env:LOCALAPPDATA\rpncalc`

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

Loading config based on environment

```python
# rpncalc/utils.py
class Config:
    ...
    def load_env(self) -> None:
        var = "RPNCALC_CONFIG_DIR"
        config_dir = os.environ.get(var)
        if not config_dir:
            return
        path = pathlib.Path(config_dir)
        ini_path = path / "rpncalc.ini"
        if not ini_path.exists():
            raise FileNotFoundError(
                f"{ini_path} not found")
        self.load(ini_path)
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

Loading config based on environment

```python
# rpncalc/utils.py
class Config:
    ...
    def load_env(self) -> None:
        var = "RPNCALC_CONFIG_DIR"
        config_dir = os.environ.get(var)
        if not config_dir:
            return
        path = pathlib.Path(config_dir)
        ini_path = path / "rpncalc.ini"
        if not ini_path.exists():
            raise FileNotFoundError(
                f"{ini_path} not found")
        self.load(ini_path)
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

Loading config based on environment

```python
# rpncalc/utils.py
class Config:
    ...
    def load_env(self) -> None:
        var = "RPNCALC_CONFIG_DIR"
        config_dir = os.environ.get(var)
        if not config_dir:
            return
        path = pathlib.Path(config_dir)
        ini_path = path / "rpncalc.ini"
        if not ini_path.exists():
            raise FileNotFoundError(
                f"{ini_path} not found")
        self.load(ini_path)
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

Loading config based on environment

```python
# rpncalc/utils.py
class Config:
    ...
    def load_env(self) -> None:
        var = "RPNCALC_CONFIG_DIR"
        config_dir = os.environ.get(var)
        if not config_dir:
            return
        path = pathlib.Path(config_dir)
        ini_path = path / "rpncalc.ini"
        if not ini_path.exists():
            raise FileNotFoundError(
                f"{ini_path} not found")
        self.load(ini_path)
```

Exercise: In `test_rpn_v2.py`, write a `TestConfig` class, containing:

- A `config_dir` fixture which uses `tmp_path` and `monkeypatch` to set `RPNCALC_CONFIG_DIR` to a temp directory, then returns the `Path`.
- Two tests for `config.load_env()`, using that fixture as an argument:
  - Creating a `rpncalc.ini` ⇒ prompt value should change
  - Keeping the directory empty ⇒ should raise `FileNotFoundError`

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Using fixtures implicitly

autouse

If you want to prepare the environment for all tests implicitly, use `autouse=True`:

```
# fixtures/test_autouse.py
class TestEmptyHomedir:

    @pytest.fixture(autouse=True)
    def tmp_homedir(self, tmp_path, monkeypatch):
        monkeypatch.setenv('HOME', str(tmp_path))
        return tmp_path

    def test_a(self, tmp_homedir):
        assert not list(tmp_homedir.iterdir())

    def test_b(self):
        assert not list(pathlib.Path.home().iterdir())
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Using fixtures implicitly

autouse

If you want to prepare the environment for all tests implicitly, use `autouse=True`:

```
# fixtures/test_autouse.py
class TestEmptyHomedir:

    @pytest.fixture(autouse=True)
    def tmp_homedir(self, tmp_path, monkeypatch):
        monkeypatch.setenv('HOME', str(tmp_path))
        return tmp_path

    def test_a(self, tmp_homedir):
        assert not list(tmp_homedir.iterdir())

    def test_b(self):
        assert not list(pathlib.Path.home().iterdir())
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Using fixtures implicitly

autouse

If you want to prepare the environment for all tests implicitly, use `autouse=True`:

```
# fixtures/test_autouse.py
class TestEmptyHomedir:

    @pytest.fixture(autouse=True)
    def tmp_homedir(self, tmp_path, monkeypatch):
        monkeypatch.setenv('HOME', str(tmp_path))
        return tmp_path

    def test_a(self, tmp_homedir):
        assert not list(tmp_homedir.iterdir())

    def test_b(self):
        assert not list(pathlib.Path.home().iterdir())
```

The patching will be done implicitly for every test for which the fixture is visible.

We can do something at the beginning of the test run via `scope="session"`.

We still return the `tmp_path` object in case a test wants to access it.

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Using fixtures implicitly

usefixtures marker

Using fixtures without specifying them as function arguments:

```
# fixtures/test_usefixtures.py
@pytest.fixture
def tmp_homedir(tmp_path, monkeypatch):
    monkeypatch.setenv('HOME', str(tmp_path))
    return tmp_path

@pytest.mark.usefixtures("tmp_homedir")
class TestHomeDir:
    def test_empty(self):
        assert not list(pathlib.Path.home().iterdir())
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Using fixtures implicitly

Exercise

Revisit your `TestConfig` class. The second test (`FileNotFoundError` when the directory is empty) should not actually need the return value of `config_dir`, so you can:

- Remove the argument and use `autouse=True` on the fixture instead
- Remove `autouse=True` again, and use `@pytest.mark.usefixtures` on the test

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Introspecting calling site

request

Fixture functions can receive a `request` object with attributes:

```
@pytest.fixture
def myfix(request: pytest.FixtureRequest):
    request.function # test function/method
    request.cls      # class of test
    request.instance # class instance
    request.module   # module of test

    request.fspath   # path object of module
    request.node     # collection node
    request.config   # pytest config object

def test_myfix(myfix):
    ...
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Introspecting calling site

request: Accessing markers

Using the `request` fixture, you can get markers from test nodes:

```
@pytest.fixture
def server_config(request: pytest.FixtureRequest) -> ServerConfig:
    marker = request.node.get_closest_marker("server_config")
    if marker is None:
        return ServerConfig()
    return ServerConfig(*marker.args, **marker.kwargs)
```

The returned `Mark` object has `args` and `kwargs` attributes with the arguments passed to it. With `@pytest.mark.server_config("production.json", strict=True)`:

```
marker.args    →  ["production.json"]
marker.kwargs  →  {"strict": True}
```

Exercise: Write a fixture whose returned value depends on the presence of a marker on the test function.

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Configuring fixtures with command line options

Add and use command line options:

```
# fixtures/cli_opt/conftest.py
def pytest_addoption(parser: pytest.Parser) -> None:
    parser.addoption("--server-ip", type=str)

@pytest.fixture
def server_ip(request: pytest.FixtureRequest) -> str:
    return request.config.option.server_ip
```

Exercise:

- Use this `conftest.py`, write a test using `server_ip` fixture
- Invoke with different `--server-ip` option values

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Parametrizing fixtures

```python
# fixtures/test_fixture_param.py

@pytest.fixture(params=[
    ">",
    "rpn>",
])
def config(request):
    c = Config(prompt=request.param)
    return c

def test_init(config):
    print(config.prompt)
    assert False
```

Now each test using the `config` fixture will run twice!

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Indirect parametrization

```python
# fixtures/test_parametrize_indirect.py

@pytest.fixture
def config(request):
    c = Config(prompt=request.param)
    return c

@pytest.mark.parametrize(
    "config, length", [
        (">", 1), ("(rpn)", 5),
    ],
    indirect=["config"],
)
def test_prompt(config: Config, length: int):
    assert len(config.prompt) == length
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Indirect parametrization

```python
# fixtures/test_parametrize_indirect.py

@pytest.fixture
def config(request):
    c = Config(prompt=request.param)
    return c

@pytest.mark.parametrize(
    "config, length", [
        (">", 1), ("(rpn)", 5),
    ],
    indirect=["config"],
)
def test_prompt(config: Config, length: int):
    assert len(config.prompt) == length
```

Can take a list of argument names (like `indirect=["config"]` here), or `indirect=True` to make all indirect.

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Debugging failing tests

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Arguments for debugging test issues

	`--tb`	Control traceback generation `--tb=auto` / `long` / `short` / `line` / `native` / `no`
`-l`	`--showlocals`	Show locals in tracebacks
`--lf`	`--last-failed`	Run last-failed only
`--ff`	`--failed-first`	Run last-failed first
`--nf`	`--new-first`	Run new test files first
`--sw`	`--stepwise`	Look at failures step by step
`-x`	`--maxfail=`*n*	Exit instantly on first / *n*-th failure
	`--pdb`	Drop into Python debugger on failures
	`--trace`	Drop into Python debugger for every test
	`--durations=`*n*	Show the *n* slowest tests/fixtures

See `pytest -h` (`--help`) for many more options.

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Tracing fixture setup/teardown

`--setup-show` Show fixtures as they are set up, used and torn down.

`--setup-only` Only setup fixtures, do not execute tests.

`--setup-plan` Show what fixtures/tests would be executed, but don't run.

Output:

```
fixtures/test_fixture.py
    SETUP    F rpn
    fixtures/test_fixture.py::test_empty_stack (fixtures used: rpn) .
    TEARDOWN F rpn
```

F: function scope, there is also **C**lass, **M**odule, **P**ackage, and **S**ession

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Adding information to an assert

Using a comma after `assert [...]`, additional information can be printed:

```
def test_add(rpn: RPNCalculator):
    rpn.evaluate("2")
    rpn.evaluate("3")
    rpn.evaluate("1")
    rpn.evaluate("+")
    assert rpn.stack[-1] == 5, rpn.stack
```

Output:

```
>           assert rpn.stack[-1] == 5, rpn.stack
E     AssertionError: [2.0, 4.0]
E     assert 4.0 == 5
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Using pdb for debugging

pdb is a command-line debugger for Python.
Trigger it with `--pdb` or `--trace` in pytest, or `breakpoint()` in your code.

At the `(pdb)` prompt, you can use:

`bt` / `w` / `where`	Print the traceback
`l` / `list`	Show the current source code
`h` / `help`	Show help
`p` / `pp`	(Pretty) print a variable
`c` / `continue`	Continue to next breakpoint
`d` / `down`, `u` / `up`	Move up/down the stack
`interact`	Open Python shell

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Showing slow test durations

Running e.g. `pytest --durations=20` reveals slow tests:

```
2.00s setup    fixtures/test_fixture_scope_reset.py::test_a
2.00s setup    fixtures/test_fixture_scope.py::test_b
2.00s setup    fixtures/test_fixture_scope.py::test_a
1.00s call     marking/test_marking.py::test_slow_api
0.27s call     mocking/test_real.py::test_convert
[...]

(4 durations < 0.005s hidden.  Use -vv to show these durations.)
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Mocking

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

You...

- Used **unittest.mock** or **pytest-mock**?

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

You...

- Used **unittest.mock** or **pytest-mock**?
- Used a mocking lib like **responses**, **freezegun**, **time-machine**?

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

You...

- Used **unittest.mock** or **pytest-mock**?
- Used a mocking lib like **responses**, **freezegun**, **time-machine**?
- Used **VCR.py**?

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

You...

- Used **unittest.mock** or **pytest-mock**?
- Used a mocking lib like **responses**, **freezegun**, **time-machine**?
- Used **VCR.py**?
- Used **pytest-httpserver** or similar?

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

You...

- Used **unittest.mock** or **pytest-mock**?
- Used a mocking lib like **responses**, **freezegun**, **time-machine**?
- Used **VCR.py**?
- Used **pytest-httpserver** or similar?
- Used **hypothesis**?

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Mocking: injecting “fake” objects

“Mocking out” dependencies helps to:

- Avoid depending on external resources
- Avoid depending on (network) IO
- Avoid depending on heavy integration with other code
- Have smaller, faster tests focused on one aspect

However, mocking increases test maintenance costs.

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

Adding currency conversion

`requests` Simple HTTP client library for Python

`exchangerate.host` Free currency conversion API, no API key needed (thanks!)

```
>>> import requests
>>> url = "https://api.exchangerate.host/latest"
>>> res = requests.get(url)
>>> res.json()
{
    'motd': {'msg': ..., 'url': ...},
    'success': True,
    'base': 'EUR',
    'date': '2023-04-25',
    'rates': {..., 'CHF': 0.980457, ...}
}
```

Requests
http for humans

exchangerate.host

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

Adding currency conversion

In `code/`, using `python -m rpncalc.rpn_v3`:

```
> 100
> eur2chf
Fetching exchange rates...
98.05
> 10
> chf2eur
10.20
> q
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

Code changes

```python
# rpncalc/rpn_v3.py

from rpncalc.utils import calc, Config
from rpncalc.convert import Converter

class RPNCalculator:

    def __init__(self, config):
        self.converter = Converter()
        self.config = config
        self.stack = []
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

Code changes

```
# rpncalc/rpn_v3.py

class RPNCalculator:
    ...
    def evaluate(self, inp):
        try:
            self.stack.append(float(inp))
            return
        except ValueError:
            pass

        if inp in ["eur2chf", "chf2eur"]:
            self._evaluate_convert(inp)
            return

        ...
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

Code changes

```
    def _evaluate_convert(self, inp):
        try:
            amount = self.stack.pop()
        except IndexError:
            print('Not enough operands')
            return

        if inp == "eur2chf":
            res = self.converter.eur2chf(amount)
        else:
            assert inp == "chf2eur", inp
            res = self.converter.chf2eur(amount)

        self.stack.append(res)
        print(f"{res:.2f}")
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

Code changes

```
    def _evaluate_convert(self, inp):
        try:
            amount = self.stack.pop()
        except IndexError:
            print('Not enough operands')
            return

        if inp == "eur2chf":
            res = self.converter.eur2chf(amount)
        else:
            assert inp == "chf2eur", inp
            res = self.converter.chf2eur(amount)

        self.stack.append(res)
        print(f"{res:.2f}")
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

Code changes

```
    def _evaluate_convert(self, inp):
        try:
            amount = self.stack.pop()
        except IndexError:
            print('Not enough operands')
            return

        if inp == "eur2chf":
            res = self.converter.eur2chf(amount)
        else:
            assert inp == "chf2eur", inp
            res = self.converter.chf2eur(amount)

        self.stack.append(res)
        print(f"{res:.2f}")
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

Code changes

```python
# rpncalc/convert.py

class Converter:

    def __init__(self) -> None:
        self._eur2chf_rate: float | None = None

    def eur2chf(self, amount: float) -> float:
        # Fetch if needed, multiply amount

    def chf2eur(self, amount: float) -> float:
        # Fetch if needed, divide amount

    def _fetch(self) -> float:
        # Download and return rate
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the v3 calculator

Functional test

A functional test might look like this:

```python
@pytest.fixture
def rpn() -> RPNCalculator:
    return RPNCalculator(Config())

def test_convert(rpn: RPNCalculator):
    rpn.stack = [10]
    rpn.evaluate("eur2chf")
    rpn.evaluate("chf2eur")
    assert rpn.stack[-1] == 10
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the v3 calculator

Functional test

A functional test might look like this:

```python
@pytest.fixture
def rpn() -> RPNCalculator:
    return RPNCalculator(Config())

def test_convert(rpn: RPNCalculator):
    rpn.stack = [10]
    rpn.evaluate("eur2chf")
    rpn.evaluate("chf2eur")
    assert rpn.stack[-1] == 10
```

Observations:

+ Testing the real API, like a user would
+ Fixture is very simple
– Depends on reachability of remote HTTP API server
– Somewhat slow, depending on network
– Hard to test actual values

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the v3 calculator

Fake class

```python
# mocking/test_fake.py

class FakeConverter:

    RATE = 2

    def eur2chf(self, amount: float) -> float:
        return amount * self.RATE

    def chf2eur(self, amount: float) -> float:
        return amount / self.RATE
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the v3 calculator

Fixture and test

```python
# mocking/test_fake.py

@pytest.fixture
def rpn(monkeypatch: pytest.MonkeyPatch) -> RPNCalculator:
    rpn = RPNCalculator(Config())
    monkeypatch.setattr(rpn, "converter", FakeConverter())
    return rpn

def test_convert(rpn: RPNCalculator):
    rpn.stack = [10]
    rpn.evaluate("eur2chf")
    assert rpn.stack == [20]
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the v3 calculator

Result

Observations:

+ Faster
+ No network dependency
+ Test is more minimal: Can test EUR ↔ CHF independently, with real values
– More complexity in fixture
– Patching requires care: If we e.g. change `self.converter` to `self._converter`, test code needs changes too.

How can we do better?

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the v3 calculator

Result

Observations:

+ Faster
+ No network dependency
+ Test is more minimal: Can test EUR ↔ CHF independently, with real values
– More complexity in fixture
– Patching requires care: If we e.g. change `self.converter` to `self._converter`, test code needs changes too.

How can we do better? Dependency injection!

Demo:

- Adjust `RPNCalculator` in `rpncalc/rpn_v3.py`, so that it takes a `converter` argument and stores it in `self.converter`
- Adjust the `if __name__ == "__main__":` block, to create and pass a `Converter()`, like with `Config()` (use keyword arguments?)
- Make sure running the calculator still works (`python -m rpncalc.rpn_v3`)
- Adjust the `rpn` fixture in `mocking/test_fake.py` to pass `FakeConverter()` as argument instead of using `monkeypatch`

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the v3 calculator

Ensuring correctness

How do we ensure our `FakeConverter` behaves like the real one?

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the v3 calculator

Ensuring correctness

How do we ensure our `FakeConverter` behaves like the real one?

Various options:

- Use a library like `unittest.mock` with `spec` / `autospec=True` (weak guarantee)
- Use a type checker like `mypy` to ensure the interface matches (stronger guarantee)
- Periodically run tests with the real converter instead (strong guarantee)

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the v3 calculator

Using unittest.mock instead

```python
# test_fake_mock.py

@pytest.fixture
def rpn(mocker: pytest_mock.MockerFixture) -> RPNCalculator:
    mock = mocker.Mock(spec=Converter)
    mock.eur2chf.return_value = 20
    mock.chf2eur.return_value = 5
    return RPNCalculator(config=Config(), converter=mock)

def test_convert(rpn: RPNCalculator):
    rpn.stack = [10]
    rpn.evaluate("eur2chf")
    assert rpn.stack == [20]

    rpn.converter.eur2chf.assert_called_once_with(10)
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the v3 calculator

Using unittest.mock instead

```python
# test_fake_mock.py

@pytest.fixture
def rpn(mocker: pytest_mock.MockerFixture) -> RPNCalculator:
    mock = mocker.Mock(spec=Converter)
    mock.eur2chf.return_value = 20
    mock.chf2eur.return_value = 5
    return RPNCalculator(config=Config(), converter=mock)

def test_convert(rpn: RPNCalculator):
    rpn.stack = [10]
    rpn.evaluate("eur2chf")
    assert rpn.stack == [20]

    rpn.converter.eur2chf.assert_called_once_with(10)
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the v3 calculator

Using unittest.mock instead

```python
# test_fake_mock.py

@pytest.fixture
def rpn(mocker: pytest_mock.MockerFixture) -> RPNCalculator:
    mock = mocker.Mock(spec=Converter)
    mock.eur2chf.return_value = 20
    mock.chf2eur.return_value = 5
    return RPNCalculator(config=Config(), converter=mock)

def test_convert(rpn: RPNCalculator):
    rpn.stack = [10]
    rpn.evaluate("eur2chf")
    assert rpn.stack == [20]

    rpn.converter.eur2chf.assert_called_once_with(10)
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the v3 calculator

Using unittest.mock instead

```python
# test_fake_mock.py

@pytest.fixture
def rpn(mocker: pytest_mock.MockerFixture) -> RPNCalculator:
    mock = mocker.Mock(spec=Converter)
    mock.eur2chf.return_value = 20
    mock.chf2eur.return_value = 5
    return RPNCalculator(config=Config(), converter=mock)

def test_convert(rpn: RPNCalculator):
    rpn.stack = [10]
    rpn.evaluate("eur2chf")
    assert rpn.stack == [20]

    rpn.converter.eur2chf.assert_called_once_with(10)
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the v3 calculator

Using type checking

```
class AbstractConverter:
    _eur2chf_rate: float | None

    def eur2chf(self, amount: float) -> float:
        raise NotImplementedError

    def chf2eur(self, amount: float) -> float:
        raise NotImplementedError

    def _fetch(self) -> float:
        raise NotImplementedError
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the v3 calculator

Using type checking

```
class AbstractConverter:
    _eur2chf_rate: float | None

    def eur2chf(self, amount: float) -> float:
        raise NotImplementedError

    def chf2eur(self, amount: float) -> float:
        raise NotImplementedError

    def _fetch(self) -> float:
        raise NotImplementedError

# rpncalc/convert.py
class Converter(AbstractConverter):
    ...
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the v3 calculator

Using type checking

```
class AbstractConverter:
    _eur2chf_rate: float | None

    def eur2chf(self, amount: float) -> float:
        raise NotImplementedError

    def chf2eur(self, amount: float) -> float:
        raise NotImplementedError

    def _fetch(self) -> float:
        raise NotImplementedError
```

```
# rpncalc/convert.py
class Converter(AbstractConverter):
    ...
```

```
# tests/test_fake.py
class FakeConverter(AbstractConverter):
    ...
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the v3 calculator

Testing against real API

```python
# conftest.py

def pytest_addoption(parser: pytest.Parser) -> None:
    parser.addoption("--real-api", action="store_true")

@pytest.fixture
def rpn(request: pytest.FixtureRequest) -> RPNCalculator:
    if request.config.option.real_api:
        converter = Converter()
    else:
        converter = FakeConverter()
    return RPNCalculator(config=Config(), converter=converter)
```

`$ pytest` ⇒ `FakeConverter()` `$ pytest --real-api` ⇒ real `Converter()`

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the v3 calculator

Where we are, what's next

We've found various strategies to test the `RPNCalculator` changes:

- Using `monkeypatch` with a hand-written fake class
- Using dependency injection, avoiding need to patch
- Using `unittest.mock` instead via `pytest-mock`
- Ensuring correctness on a type level with an abstract class and e.g. `mypy`
- Ensuring correctness by periodically running tests against the real API

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the v3 calculator

Where we are, what's next

We've found various strategies to test the `RPNCalculator` changes:

- Using `monkeypatch` with a hand-written fake class
- Using dependency injection, avoiding need to patch
- Using `unittest.mock` instead via `pytest-mock`
- Ensuring correctness on a type level with an abstract class and e.g. `mypy`
- Ensuring correctness by periodically running tests against the real API

We still need to test the `Converter` class itself.

Next up:

- Naive approach: Using `monkeypatch` to patch `requests.get`
- Using `unittest.mock` Still low-level!
- Using mocking libraries: `responses`, `VCR.py`
- Back to functional tests: `pytest-httpserver`

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

The Converter class

```python
# rpncalc/convert.py
import requests

class Converter:

    API_URL = "https://api.exchangerate.host/convert"
    USER_AGENT = "rpncalc/0.1 (florian@bruhin.software)"

    def __init__(self) -> None:
        self._eur2chf_rate = None

    def eur2chf(self, amount: float) -> float:
        if self._eur2chf_rate is None:
            self._eur2chf_rate = self._fetch()
        return amount * self._eur2chf_rate
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

The Converter class

```python
# rpncalc/convert.py
import requests

class Converter:

    API_URL = "https://api.exchangerate.host/convert"
    USER_AGENT = "rpncalc/0.1 (florian@bruhin.software)"

    def __init__(self) -> None:
        self._eur2chf_rate = None

    def eur2chf(self, amount: float) -> float:
        if self._eur2chf_rate is None:
            self._eur2chf_rate = self._fetch()
        return amount * self._eur2chf_rate
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

The Converter class

```python
# rpncalc/convert.py
import requests

class Converter:

    API_URL = "https://api.exchangerate.host/convert"
    USER_AGENT = "rpncalc/0.1 (florian@bruhin.software)"

    def __init__(self) -> None:
        self._eur2chf_rate = None

    def eur2chf(self, amount: float) -> float:
        if self._eur2chf_rate is None:
            self._eur2chf_rate = self._fetch()
        return amount * self._eur2chf_rate
```

```python
def chf2eur(self, amount):
    # ... same code ...
    #
    return amount / ...
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

The Converter class

```python
# rpncalc/convert.py
class Converter:
    ...
    def _fetch(self) -> float:
        print("Fetching exchange rates...")
        params = {"from": "EUR", "to": "CHF"}
        headers = {"User-Agent": self.USER_AGENT}
        response = requests.get(
            self.API_URL,
            params=params,
            headers=headers)
        response.raise_for_status()
        data = response.json()
        assert data["success"], data
        return data["result"]
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reverse Polish Notation (RPN)

The Converter class

```python
# rpncalc/convert.py
class Converter:
    ...
    def _fetch(self) -> float:
        print("Fetching exchange rates...")
        params = {"from": "EUR", "to": "CHF"}
        headers = {"User-Agent": self.USER_AGENT}
        response = requests.get(
            self.API_URL,
            params=params,
            headers=headers)
        response.raise_for_status()
        data = response.json()
        assert data["success"], data
        return data["result"]
```


```json
{
  "motd": { ... },
  "success": true,
  "query": {
    "from": "EUR",
    "to": "CHF",
    "amount": 1
  },
  "info": {
    "rate": 0.980457
  },
  "historical": false,
  "date": "2023-04-25",
  "result": 0.980457
}
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the Converter

Initial approach: monkeypatch

```python
# rpncalc/convert.py
class Converter:
    ...
    def _fetch(self) -> float:
        ...
        response = requests.get(...)
        response.raise_for_status()
        data = response.json()
        assert data["success"], data
        return data["result"]
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the Converter

Initial approach: monkeypatch

```python
# rpncalc/convert.py
class Converter:
    ...
    def _fetch(self) -> float:
        ...
        response = requests.get(...)
        response.raise_for_status()
        data = response.json()
        assert data["success"], data
        return data["result"]
```

```python
# mocking/converter/
# test_monkeypatch.py

class FakeResponse:

    def __init__(self, data):
        self._data = data

    def raise_for_status(self):
        pass

    def json(self):
        return self._data
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the Converter

Initial approach: monkeypatch

```python
# mocking/converter/test_monkeypatch.py

def fake_get(url: str, params: dict, headers: dict) -> FakeResponse:
    assert url == Converter.API_URL
    assert params == {"from": "EUR", "to": "CHF"}
    assert headers == {"User-Agent": Converter.USER_AGENT}
    return FakeResponse({"success": True, "result": 2})
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the Converter

Initial approach: monkeypatch

```python
# mocking/converter/test_monkeypatch.py

def fake_get(url: str, params: dict, headers: dict) -> FakeResponse:
    assert url == Converter.API_URL
    assert params == {"from": "EUR", "to": "CHF"}
    assert headers == {"User-Agent": Converter.USER_AGENT}
    return FakeResponse({"success": True, "result": 2})

@pytest.fixture(autouse=True)
def patch_requests_get(monkeypatch: pytest.MonkeyPatch) -> None:
    monkeypatch.setattr(requests, "get", fake_get)
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the Converter

Initial approach: monkeypatch

```python
# mocking/converter/test_monkeypatch.py


@pytest.fixture
def converter() -> Converter:
    return Converter()


def test_eur2chf(converter: Converter):
    assert converter.eur2chf(1) == 2

def test_chf2eur(converter: Converter):
    assert converter.chf2eur(1) == 0.5
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the Converter

Initial approach: monkeypatch

```python
# mocking/converter/test_monkeypatch.py

@pytest.fixture
def converter() -> Converter:
    return Converter()


def test_eur2chf(converter: Converter):
    assert converter.eur2chf(1) == 2

def test_chf2eur(converter: Converter):
    assert converter.chf2eur(1) == 0.5
```

Observations:

+ Testing `Converter` without network access (fast and isolated)
+ Tests are very simple and precise

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the Converter

Initial approach: monkeypatch

```python
# mocking/converter/test_monkeypatch.py

@pytest.fixture
def converter() -> Converter:
    return Converter()


def test_eur2chf(converter: Converter):
    assert converter.eur2chf(1) == 2

def test_chf2eur(converter: Converter):
    assert converter.chf2eur(1) == 0.5
```

Observations:

+ Testing `Converter` without network access (fast and isolated)
+ Tests are very simple and precise

– Behind the scenes: Lots of complexity!
– If the implementation changes (e.g. using `requests.Session`), the tests need changes too!

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the Converter

Reducing complexity: unittest.mock?

```python
@pytest.fixture(autouse=True)
def mock_requests_get(mocker: pytest_mock.MockerFixture):
    data = {"success": True, "result": 2}
    mock_get = mocker.patch.object(requests, "get", autospec=True)
    mock_get(Converter.API_URL).json.return_value = data
    yield mock_get
    mock_get.assert_called_with(
        Converter.API_URL,
        params={"from": "EUR", "to": "CHF"},
        headers={"User-Agent": Converter.USER_AGENT})
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the Converter

Reducing complexity: unittest.mock?

```python
@pytest.fixture(autouse=True)
def mock_requests_get(mocker: pytest_mock.MockerFixture):
    data = {"success": True, "result": 2}
    mock_get = mocker.patch.object(requests, "get", autospec=True)
    mock_get(Converter.API_URL).json.return_value = data
    yield mock_get
    mock_get.assert_called_with(
        Converter.API_URL,
        params={"from": "EUR", "to": "CHF"},
        headers={"User-Agent": Converter.USER_AGENT})
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the Converter

Reducing complexity: unittest.mock?

```python
@pytest.fixture(autouse=True)
def mock_requests_get(mocker: pytest_mock.MockerFixture):
    data = {"success": True, "result": 2}
    mock_get = mocker.patch.object(requests, "get", autospec=True)
    mock_get(Converter.API_URL).json.return_value = data
    yield mock_get
    mock_get.assert_called_with(
        Converter.API_URL,
        params={"from": "EUR", "to": "CHF"},
        headers={"User-Agent": Converter.USER_AGENT})
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the Converter

Reducing complexity: unittest.mock?

```python
@pytest.fixture(autouse=True)
def mock_requests_get(mocker: pytest_mock.MockerFixture):
    data = {"success": True, "result": 2}
    mock_get = mocker.patch.object(requests, "get", autospec=True)
    mock_get(Converter.API_URL).json.return_value = data
    yield mock_get
    mock_get.assert_called_with(
        Converter.API_URL,
        params={"from": "EUR", "to": "CHF"},
        headers={"User-Agent": Converter.USER_AGENT})
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the Converter

Reducing complexity: unittest.mock?

```
@pytest.fixture(autouse=True)
def mock_requests_get(mocker: pytest_mock.MockerFixture):
    data = {"success": True, "result": 2}
    mock_get = mocker.patch.object(requests, "get", autospec=True)
    mock_get(Converter.API_URL).json.return_value = data
    yield mock_get
    mock_get.assert_called_with(
        Converter.API_URL,
        params={"from": "EUR", "to": "CHF"},
        headers={"User-Agent": Converter.USER_AGENT})
```

Observations:

+ No need to hand-roll a `FakeResponse`
+ More safety due to `autospec=True`

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the Converter

Reducing complexity: unittest.mock?

```
@pytest.fixture(autouse=True)
def mock_requests_get(mocker: pytest_mock.MockerFixture):
    data = {"success": True, "result": 2}
    mock_get = mocker.patch.object(requests, "get", autospec=True)
    mock_get(Converter.API_URL).json.return_value = data
    yield mock_get
    mock_get.assert_called_with(
        Converter.API_URL,
        params={"from": "EUR", "to": "CHF"},
        headers={"User-Agent": Converter.USER_AGENT})
```

Observations:

+ No need to hand-roll a `FakeResponse`
+ More safety due to `autospec=True`

– Tedious; "weird" `unittest.mock` API
– Still needs to know about implementation

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Maybe there's no need to hand-roll mocks?

Ready-made mocking libraries

requests	**responses**, betamax, requests-mock
aiohttp	aioresponses
HTTP in general	**VCR.py**, HTTPretty, pook
sockets in general	mocket
time/datetime	freezegun, time-machine

For most of those, there's a corresponding pytest plugin, making integration easier.

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Maybe there's no need to hand-roll mocks?

Ready-made mocking libraries

requests	**responses**, betamax, requests-mock
aiohttp	aioresponses
HTTP in general	**VCR.py**, HTTPretty, pook
sockets in general	mocket
time/datetime	freezegun, time-machine

For most of those, there's a corresponding pytest plugin, making integration easier.

Result: More high-level abstraction for mocking (e.g. HTTP responses instead of the Python API).

Well-tested mocks, usually covering the entire API of a library.

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the Converter

High-level mocking with responses

```python
# mocking/converter/test_responses.py
from responses import RequestsMock, matchers

@pytest.fixture(autouse=True)
def patch_requests_get(responses: RequestsMock) -> None:
    params = {"from": "EUR", "to": "CHF"}
    headers = {"User-Agent": Converter.USER_AGENT}
    responses.get(
        Converter.API_URL,
        json={"success": True, "result": 2},
        match=[
            matchers.query_param_matcher(params),
            matchers.header_matcher(headers),
        ],
    )
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the Converter

High-level mocking with responses

```python
# mocking/converter/test_responses.py
from responses import RequestsMock, matchers

@pytest.fixture(autouse=True)
def patch_requests_get(responses: RequestsMock) -> None:
    params = {"from": "EUR", "to": "CHF"}
    headers = {"User-Agent": Converter.USER_AGENT}
    responses.get(
        Converter.API_URL,
        json={"success": True, "result": 2},
        match=[
            matchers.query_param_matcher(params),
            matchers.header_matcher(headers),
        ],
    )
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the Converter

High-level mocking with responses

```python
# mocking/converter/test_responses.py
from responses import RequestsMock, matchers

@pytest.fixture(autouse=True)
def patch_requests_get(responses: RequestsMock) -> None:
    params = {"from": "EUR", "to": "CHF"}
    headers = {"User-Agent": Converter.USER_AGENT}
    responses.get(
        Converter.API_URL,
        json={"success": True, "result": 2},
        match=[
            matchers.query_param_matcher(params),
            matchers.header_matcher(headers),
        ],
    )
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the Converter

High-level mocking with responses

```python
# mocking/converter/test_responses.py
from responses import RequestsMock, matchers

@pytest.fixture(autouse=True)
def patch_requests_get(responses: RequestsMock) -> None:
    params = {"from": "EUR", "to": "CHF"}
    headers = {"User-Agent": Converter.USER_AGENT}
    responses.get(
        Converter.API_URL,
        json={"success": True, "result": 2},
        match=[
            matchers.query_param_matcher(params),
            matchers.header_matcher(headers),
        ],
    )
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the Converter

High-level mocking with responses

```
responses.get(
    Converter.API_URL,
    json={
        "success": True,
        "result": 2
    },
    match=[...],
)
```

Observations:

+ Much more high-level API
+ Easy to use, easy to validate requests
+ Covers the entire API of `requests`

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the Converter

High-level mocking with responses

```
responses.get(
    Converter.API_URL,
    json={
        "success": True,
        "result": 2
    },
    match=[...],
)
```

Observations:

+ Much more high-level API
+ Easy to use, easy to validate requests
+ Covers the entire API of `requests`
± We still need to provide the data manually
  + Possibility to test corner cases
  – Needs care so data is same as real API

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the Converter

High-level mocking with responses

```
responses.get(
    Converter.API_URL,
    json={
        "success": True,
        "result": 2
    },
    match=[...],
)
```

Observations:

+ Much more high-level API
+ Easy to use, easy to validate requests
+ Covers the entire API of `requests`
± We still need to provide the data manually
  + Possibility to test corner cases
  − Needs care so data is same as real API

Not many high-level mocking libraries exist (wish more projects would do this!)
But if one does, **use it**!

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the Converter

High(er)-level mocking with VCR.py

```python
# mocking/converter/test_vcr.py

@pytest.mark.vcr
def test_eur2chf(converter: Converter):
    assert converter.eur2chf(1) == pytest.approx(0.98, rel=1e-2)

@pytest.mark.vcr
def test_chf2eur(converter: Converter):
    assert converter.chf2eur(1) == pytest.approx(1.02, rel=1e-2)
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the Converter

High(er)-level mocking with VCR.py

```python
# mocking/converter/test_vcr.py

@pytest.mark.vcr
def test_eur2chf(converter: Converter):
    assert converter.eur2chf(1) == pytest.approx(0.98, rel=1e-2)

@pytest.mark.vcr
def test_chf2eur(converter: Converter):
    assert converter.chf2eur(1) == pytest.approx(1.02, rel=1e-2)
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the Converter

High(er)-level mocking with VCR.py

```python
# mocking/converter/test_vcr.py

@pytest.mark.vcr
def test_eur2chf(converter: Converter):
    assert converter.eur2chf(1) == pytest.approx(0.98, rel=1e-2)
```

```
$ pytest test_vcr.py
[...]
vcr.errors.CannotOverwriteExistingCassetteException: Can't overwrite
    existing cassette ('.../converter/cassettes/test_vcr/
    test_eur2chf.yaml') in your current record mode ('none').
No match for the request (<Request (GET) https://api.exchangerate.host/
    convert?from=EUR&to=CHF>) was found.
No similar requests, that have not been played, found.
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the Converter

High(er)-level mocking with VCR.py

```
$ pytest --record-mode=once test_vcr.py
[...]

converter/test_vcr.py ..

========== 2 passed in 0.86s ===========
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the Converter

High(er)-level mocking with VCR.py

```
$ pytest --record-mode=once test_vcr.py
[...]

converter/test_vcr.py ..

========== 2 passed in 0.86s ===========
```

```
code/mocking/converter/
├── ...
├── test_vcr.py
└── cassettes/
    └── test_vcr/
        ├── test_eur2chf.yaml
        └── test_chf2eur.yaml
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the Converter

High(er)-level mocking with VCR.py

```
$ pytest --record-mode=once test_vcr.py
[...]

converter/test_vcr.py ..

========== 2 passed in 0.86s ===========

$ pytest test_vcr.py
[...]

converter/test_vcr.py ..

========== 2 passed in 0.05s ===========
```

```
code/mocking/converter/
├── ...
├── test_vcr.py
└── cassettes/
    └── test_vcr/
        ├── test_eur2chf.yaml
        └── test_chf2eur.yaml
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the Converter

High(er)-level mocking with VCR.py

```
interactions:
- request:
    uri: https://api.exchangerate.host/convert?from=EUR&to=CHF
    method: GET
    headers:
      [...]
      User-Agent:
      - rpncalc/0.1 (florian@bruhin.software)
    body: null
  response:
    body: '{..., "success": true, "result": 0.978521}'    *
    headers: [...]
    status: {"code": 200, "message": "OK"}
version: 1
```

* body is actually stored as gzip binary, exactly as sent by the server. Shown as text here for simplicity.

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the Converter

High(er)-level mocking with VCR.py

Observations:

+ Even higher-level: No need for explicit mocking
+ Guaranteed to have a 1:1 recording of server response
+ Covers most Python HTTP APIs!
+ Easy to update cassettes if API changes server-side

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the Converter

High(er)-level mocking with VCR.py

Observations:

+ Even higher-level: No need for explicit mocking
+ Guaranteed to have a 1:1 recording of server response
+ Covers most Python HTTP APIs!
+ Easy to update cassettes if API changes server-side

– Much harder to test corner cases
(though we can write manual cassettes)

– Tests are less expressive for documentation

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Maybe there's no need to mock?

Consider using the real thing

Databases sqlite, pytest-postgresql, pytest-mongo, pytest-mysql, pytest-dynamodb

HTTP servers **pytest-httpserver**, pytest-localserver; Flask and cheroot

Others pytest-rabbitmq, pytest-solr, pytest-elasticsearch, pytest-redis, pytest-grpc, pytest-localserver (SMTP)

Generic pytest-xprocess, pytest-docker

SQLite docker Flask web development, one drop at a time

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the Converter

Using a real HTTP server

```python
# mocking/converter/test_httpserver.py
@pytest.fixture
def server_url(httpserver: pytest_httpserver.HTTPServer):
    url_path = urllib.parse.urlparse(Converter.API_URL).path

    req = httpserver.expect_request(
        url_path,
        query_string={"from": "EUR", "to": "CHF"},
        headers={"User-Agent": Converter.USER_AGENT},
    )
    req.respond_with_json({"success": True, "result": 2})

    yield httpserver.url_for(url_path)

    httpserver.check()
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the Converter

Using a real HTTP server

```python
# mocking/converter/test_httpserver.py
@pytest.fixture
def server_url(httpserver: pytest_httpserver.HTTPServer):
    url_path = urllib.parse.urlparse(Converter.API_URL).path

    req = httpserver.expect_request(
        url_path,
        query_string={"from": "EUR", "to": "CHF"},
        headers={"User-Agent": Converter.USER_AGENT},
    )
    req.respond_with_json({"success": True, "result": 2})

    yield httpserver.url_for(url_path)

    httpserver.check()
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the Converter

Using a real HTTP server

```python
# mocking/converter/test_httpserver.py
@pytest.fixture
def server_url(httpserver: pytest_httpserver.HTTPServer):
    url_path = urllib.parse.urlparse(Converter.API_URL).path

    req = httpserver.expect_request(
        url_path,
        query_string={"from": "EUR", "to": "CHF"},
        headers={"User-Agent": Converter.USER_AGENT},
    )
    req.respond_with_json({"success": True, "result": 2})

    yield httpserver.url_for(url_path)

    httpserver.check()
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the Converter

Using a real HTTP server

```python
# mocking/converter/test_httpserver.py
@pytest.fixture
def server_url(httpserver: pytest_httpserver.HTTPServer):
    url_path = urllib.parse.urlparse(Converter.API_URL).path

    req = httpserver.expect_request(
        url_path,
        query_string={"from": "EUR", "to": "CHF"},
        headers={"User-Agent": Converter.USER_AGENT},
    )
    req.respond_with_json({"success": True, "result": 2})

    yield httpserver.url_for(url_path)

    httpserver.check()
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the Converter

Using a real HTTP server

```python
# mocking/converter/test_httpserver.py
@pytest.fixture
def server_url(httpserver: pytest_httpserver.HTTPServer):
    url_path = urllib.parse.urlparse(Converter.API_URL).path

    req = httpserver.expect_request(
        url_path,
        query_string={"from": "EUR", "to": "CHF"},
        headers={"User-Agent": Converter.USER_AGENT},
    )
    req.respond_with_json({"success": True, "result": 2})

    yield httpserver.url_for(url_path)

    httpserver.check()
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the Converter

Using a real HTTP server

```python
# mocking/converter/test_httpserver.py
@pytest.fixture
def server_url(httpserver: pytest_httpserver.HTTPServer):
    url_path = urllib.parse.urlparse(Converter.API_URL).path

    req = httpserver.expect_request(
        url_path,
        query_string={"from": "EUR", "to": "CHF"},
        headers={"User-Agent": Converter.USER_AGENT},
    )
    req.respond_with_json({"success": True, "result": 2})

    yield httpserver.url_for(url_path)

    httpserver.check()
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the Converter

Using a real HTTP server

```python
@pytest.fixture
def converter(
    server_url: str,  # e.g. http://localhost:41475/convert
    monkeypatch: pytest.MonkeyPatch,
) -> Converter:
    conv = Converter()
    monkeypatch.setattr(conv, "API_URL", server_url)
    return conv

def test_eur2chf(converter: Converter):
    assert converter.eur2chf(1) == 2

def test_chf2eur(converter: Converter):
    assert converter.chf2eur(1) == 0.5
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing the Converter

Using a real HTTP server

Observations:

- For things like HTTP, SMTP, ... it might be easier to use the real thing (or at least an approximation: The real API doesn't use `pytest-httpserver`)
- Various pytest plugins exist to help with this
- Double-edged sword for things that don't exist as a pure-Python library:
  - Non-Python server dependencies make tests harder to run
  - Docker is no silver bullet either
  - Many more moving parts, potentially slower or more brittle

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Different approaches to mocking

- Hand-written stubs with `monkeypatch`
  - Possibly using an abstract base class and `mypy`
  - Possibly using a `--real-thing` option to cross-check
- Automated mocks with `unittest.mock` (via `pytest-mock`)
- Write code in a flexible enough way so you don't need to patch
- Use a pre-made mocking library (e.g. `responses`, `VCR.py`)
- Can you easily use the real thing instead? (e.g. `pytest-httpserver`)

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Some plugins

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

You...

- Used **coverage**?

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

You...

- Used **coverage**?
- Know what **bdd** tests are (behavior driven development)?

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

You...

- Used **coverage**?
- Know what **bdd** tests are (behavior driven development)?
- Implemented pytest **plugin hooks**?

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Using the coverage plugin

Install plugin: `pip install pytest-cov`

Options:

`--cov=path`	filesystem path to generate coverage for
`--cov-report=type`	type of report ("term", "html", ...)
`--cov-config=path`	path of `.coveragerc` file

Demo:

- Run `pytest --cov=rpncalc/ --cov-report=term-missing rpncalc/`, take a look at the terminal output
- Rerun with `--cov-report=html`, then open `htmlcov/index.html`, look at the report

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Distributed testing

Install plugin:

```
pip install pytest-xdist
```

Run tests against multiple cores:

```
pytest -n 4
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Output plugins: pytest-instafail, pytest-rich

pytest-instafail reports failure details while tests are running, rather than the end of the test run.

pytest-rich uses the Rich library for nicer output.

```
═══ pytest session starts ═══
platform linux pytest 7.0.1 python 3.10.0
root /home/josh/pytest-rich

Collected 94 items
[100%] tests/test_bar.py ✓✗✗✗✓✗✓✗✓
[100%] tests/test_foo.py ✓✗✓✗✓✗✓✗✓✗✓✗✓✗✓✗✓✗✓✗✓✗✓✗✓✗✓✗✓✗✓✗✓✗✓✗✓✗✓✗✓✗…
[100%] tests/test_rich.py ✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓
Percent: 100%
═══ FAILURES ═══
═══ tests/test_bar.py::test_bar[1] ═══
tests/test_bar.py:8 in test_bar
i = 1

   5 # time.sleep(1)
   6 @pytest.mark.parametrize("i", range(10))
   7 def test_bar(i):
❯  8     assert i % 2 == 0
   9     if i == 2:
  10         assert 0, "Some failure"
  11     time.sleep(0.02)

8: AssertionError
❯ assert i % 2 == 0
E assert (1 % 2) == 0

tests/test_bar.py:8: AssertionError
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

pytest-bdd: Behavior driven testing

Gherkin (Cucumber)

```
# article.feature

Scenario: Publishing the article
  Given I'm an author user
  And I have an article

  When I go to the article page
  And I press the publish button

  Then no error should be shown
  And the article should be published
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

pytest-bdd: Behavior driven testing

Gherkin (Cucumber)

```
# article.feature

Scenario: Publishing the article
  Given I'm an author user
  And I have an article

  When I go to the article page
  And I press the publish button

  Then no error should be shown
  And the article should be published
```

```
# test_article.py

@when("I go to the article page")
def go_to_article(article, browser):
    browser.visit(article.url())

@when("I press the publish button")
def publish_article(browser):
    browser.find_by_id(...).click()
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Writing tests against C/C++/Rust/... libraries

To test libraries written in non-Python languages, you can:

- C++: Use `pytest-cpp` with `GoogleTest`, `Boost.Test` or `Catch2`
- Write a command line entry point and test that
- Wrap (part of) the library API in a Python extension

To wrap C libraries:

- Cython (hybrid C/Python)
- ctypes or cffi
- Classical Python C extension

For C++:

- pybind11 / nanobind
- Boost::Python
- SWIG, sip

See also: http://pytest-c-testrunner.rtfd.org/ and https://realpython.com/python-bindings-overview/

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing async code

```
async def test_asyncio():
    answer = 42
    val = await asyncio.sleep(1, result=answer)
    assert val == answer
```

`PytestUnhandledCoroutineWarning`:
`async def` functions are not natively supported and have been skipped.
You need to install a suitable plugin for your async framework, for example:

- *anyio*
- *pytest-asyncio*
- *pytest-tornasync*
- *pytest-trio*
- *pytest-twisted*

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing async code

```python
@pytest.mark.asyncio
async def test_asyncio():
    answer = 42
    val = await asyncio.sleep(1, result=answer)
    assert val == answer
```

```
======================= test session starts =========================
[...]
plugins: [...], asyncio-0.21.0, [...]
collected 1 item

plugins/test_asyncio.py::test_asyncio            PASSED          [100%]

========================= 1 passed in 1.01s ==========================
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Testing async code

```
@pytest.mark.asyncio
async def test_asyncio():
    answer = 42
    val = await asyncio.sleep(1, result=answer)
    assert val == answer
```

Strict mode:

- Async test functions need `@pytest.mark.asyncio`
- Async fixtures need `@pytest_asyncio.fixture`
- `pytest-asyncio` can coexist with other asyncio plugins

Auto mode:

- All `async def` tests/fixtures are handled automatically
- If only `asyncio` is used, no other async framework
- Opt-in, but recommended

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Plugins, plugins ...

- Property-based testing: **hypothesis**
- Customized reporting: pytest-html, pytest-slack, **pytest-rich**, **pytest-instafail**, pytest-emoji
- Repeating tests: pytest-repeat, pytest-rerunfailures, pytest-benchmark
- Framework/Language integration: pytest-twisted, pytest-django, pytest-qt, pytest-asyncio, pytest-cpp
- Coverage and mock integration: **pytest-cov**, **pytest-mock**
- Other: **pytest-bdd** (behaviour-driven testing), **pytest-xdist** (distributed testing)
- ... > 1200 more: https://docs.pytest.org/en/latest/reference/plugin_list.html

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Property-based testing with Hypothesis

Motivation

From the Hypothesis website, `hypothesis.works`:

> *Most testing is ineffective*
>
> *Normal "automated" software testing is surprisingly manual.*
> *Every scenario the computer runs, someone had to write by hand.*
> *Hypothesis can fix this.*

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Property-based testing with Hypothesis

Motivation

From the Hypothesis website, `hypothesis.works`:

> *Most testing is ineffective*
>
> *Normal "automated" software testing is surprisingly manual.*
> *Every scenario the computer runs, someone had to write by hand.*
> *Hypothesis can fix this.*

Idea behind "Property-Based testing" (popularized by QuickCheck/Haskell):

- Generate input data based on a *strategy* (how should the data look?)
- Run the test case many times (say, 100), with different random generated data
- Check for *properties* / *invariants* that should hold for all input data
- If there is a failure, *minimize* the input data to a minimal failing example

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Property-based testing with Hypothesis

Old buggy RPN calculator

```python
# rpncalc/rpn_v1.py

class RPNCalculator:
    ...

    def evaluate(self, inp):
        if inp.isdigit():
            n = float(inp)
            self.stack.append(n)
        elif inp in '+-*/':
            b = self.stack.pop()
            a = self.stack.pop()
            res = calc(a, b, inp)
            self.stack.append(res)
            print(res)
```

```
> 1
> +
Traceback (most recent call last):
  [...]
  File [...], in <module>
    rpn.run()
  File [...], in run
    self.evaluate(inp)
  File [...], in evaluate
    a = self.stack.pop()
IndexError: pop from empty list
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Property-based testing with Hypothesis

Old buggy RPN calculator: Hypothesis test

```python
# plugins/test_hypothesis_rpncalc_v1.py

from hypothesis import given, strategies as st
from rpncalc.rpn_v1 import RPNCalculator

@given(st.text())
def test_random_strings(s):
    rpn = RPNCalculator()
    rpn.evaluate(s)
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Property-based testing with Hypothesis

Old buggy RPN calculator: Hypothesis test

```python
# plugins/test_hypothesis_rpncalc_v1.py

from hypothesis import given, strategies as st
from rpncalc.rpn_v1 import RPNCalculator

@given(st.text())
def test_random_strings(s):
    rpn = RPNCalculator()
    rpn.evaluate(s)
```

```
>   b = self.stack.pop()
E   IndexError: pop from empty list
E   Falsifying example: test_random_strings(
E       s='',
E   )
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Property-based testing with Hypothesis

Old buggy RPN calculator: Hypothesis test

```python
# plugins/test_hypothesis_rpncalc_v1.py

from hypothesis import given, strategies as st
from rpncalc.rpn_v1 import RPNCalculator

@given(st.text())
def test_random_strings(s):
    rpn = RPNCalculator()
    rpn.evaluate(s)
```

```
>   b = self.stack.pop()
E   IndexError: pop from empty list
E   Falsifying example: test_random_strings(
E       s='',
E   )
```

```
elif inp in '+-*/':
    [...]
```

```
>>> '' in '+-*/'
True
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Property-based testing with Hypothesis

Surely v3 is bug-free?

```
# rpncalc/rpn_v3.py

class RPNCalculator:
    [...]

    def evaluate(self, inp):
        [...]
        try:
            b = self.stack.pop()
            a = self.stack.pop()
        except IndexError:
            print('Not enough operands')
            return
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Property-based testing with Hypothesis

Surely v3 is bug-free?

```python
# rpncalc/rpn_v3.py

class RPNCalculator:
    [...]

    def evaluate(self, inp):
        [...]
        try:
            b = self.stack.pop()
            a = self.stack.pop()
        except IndexError:
            print('Not enough operands')
            return
```

```
test_random_strings PASSED
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Property-based testing with Hypothesis

Surely v3 is bug-free?

```
# plugins/test_hypothesis_rpncalc_v3.py

@given(st.integers(), st.integers())
def test_operators(n1, n2):
    rpn = RPNCalculator(Config())
    rpn.evaluate(str(n1))
    rpn.evaluate(str(n2))
    rpn.evaluate("+")
    assert rpn.stack == [n1 + n2]
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Property-based testing with Hypothesis

Surely v3 is bug-free?

```
# plugins/test_hypothesis_rpncalc_v3.py

@given(st.integers(), st.integers())
def test_operators(n1, n2):
    rpn = RPNCalculator(Config())
    rpn.evaluate(str(n1))
    rpn.evaluate(str(n2))
    rpn.evaluate("+")
    assert rpn.stack == [n1 + n2]
E   assert [1.055531162664984 8 e+16] == [1055531162664984 9 ]
E   [...]
E   Falsifying example: test_operators(
E       n1=0,
E       n2=10555311626649849,
E   )
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Property-based testing with Hypothesis

Surely v3 is bug-free?

```
# plugins/test_hypothesis_rpncalc_v3.py

@given(st.integers(), st.integers())
def test_operators(n1, n2):
    rpn = RPNCalculator(Config())
    rpn.evaluate(str(n1))
    rpn.evaluate(str(n2))
    rpn.evaluate("+")
    assert rpn.stack == [n1 + n2]
E   assert [1.055531162664984 8 e+16] == [1055531162664984 9 ]
E   [...]
E   Falsifying example: test_operators(
E       n1=0,
E       n2=10555311626649849,
E   )
```

```
> 10555311626649849
> 0
> +
1.055531162664984 8 e+16
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Property-based testing with Hypothesis

Surely v3 is bug-free?

```
# plugins/test_hypothesis_rpncalc_v3.py

@given(st.integers(), st.integers())
def test_operators(n1, n2):
    rpn = RPNCalculator(Config())
    rpn.evaluate(str(n1))
    rpn.evaluate(str(n2))
    rpn.evaluate("+")
    assert rpn.stack == [n1 + n2]
E   assert [1.055531162664984 8 e+16] == [1055531162664984 9 ]
E   [...]
E   Falsifying example: test_operators(
E       n1=0,
E       n2=10555311626649849,
E   )
```

```
> 10555311626649849
> 0
> +
1.055531162664984 8 e+16
```

```
# rpncalc/rpn_v3.py
[...]
self.stack.append(float(inp))
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Property-based testing with Hypothesis

More sophisticated strategies

```
@given(
    st.lists(
        st.one_of(
            st.integers().map(str),
            st.floats().map(str),
            st.just("+"), st.just("-"),
            st.just("*"), st.just("/"),
        )
    )
)
def test_usage(inputs):
    rpn = RPNCalculator(Config())
    for inp in inputs:
        rpn.evaluate(inp)
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Property-based testing with Hypothesis

More sophisticated strategies

```python
@given(
    st.lists(
        st.one_of(
            st.integers().map(str),
            st.floats().map(str),
            st.just("+"), st.just("-"),
            st.just("*"), st.just("/"),
        )
    )
)
def test_usage(inputs):
    rpn = RPNCalculator(Config())
    for inp in inputs:
        rpn.evaluate(inp)
```

```
Trying example: test_usage(
  inputs=[
    '-2.2250738585e-313',
    '/', '-10', '110', '+', '+'])
Not enough operands
100.0
Not enough operands
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Property-based testing with Hypothesis

More sophisticated strategies

```python
@given(
    st.lists(
        st.one_of(
            st.integers().map(str),
            st.floats().map(str),
            st.just("+"), st.just("-"),
            st.just("*"), st.just("/"),
        )
    )
)
def test_usage(inputs):
    rpn = RPNCalculator(Config())
    for inp in inputs:
        rpn.evaluate(inp)
```

```
Trying example: test_usage(
  inputs=[
    '-2.2250738585e-313',
    '/', '-10', '110', '+', '+'])
Not enough operands
100.0
Not enough operands

Trying example: test_usage(
  inputs=['+', '+', '-', '*'],
)
Not enough operands
Not enough operands
Not enough operands
Not enough operands
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Property-based testing with Hypothesis

What it can do

- Generate data based on types/strategies
- Combine/filter various strategies to generate more sophisticated data
- Generate names, etc. via fakefactory
- Generate data matching a Django model
- Generate data matching a grammar
- Generate test code (rather than data) based on a state machine

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Writing your own plugins

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Writing pytest plugins

Plugins come in two flavours:

- Local plugins: `conftest.py` files
- Installable plugins via `setuptools` entry points

And implement one or many hooks for:

- configuration
- collection
- test running
- reporting

A hook is auto-discovered by its `pytest_···` name.

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Hook mechanism

All hook implementations have a `pytest_` prefix. The argument names **must** match the precise signature names:

ok:

```
def pytest_configure(config):
    ...
```

You can accept fewer parameters:

```
def pytest_configure():
    ...
```

mismatch:

```
def pytest_configure(cfg):
    ...
```

As with fixtures, imagine pytest calling your function with keyword arguments:

```
pytest_configure(config=pytest.Config(...))
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Anatomy of a pytest run

- `main()`
  - Import and register built-in plugins:
    `pytest_plugin_registered`
  - `pytest_addoption`
  - `pytest_cmdline_parse`
  - `pytest_cmdline_main`
    - `pytest_configure`
    - `pytest_sessionstart`
      - `pytest_report_header`
    - `pytest_collection`
      - ...
    - `pytest_runtestloop`
      - ...
    - `pytest_sessionfinish`
      - `pytest_terminal_summary`
    - `pytest_unconfigure`

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

The pytest_addoption hook

```python
def pytest_addoption(
    parser: Parser,
    pluginmanager: PytestPluginManager,
) -> None:
    """Register argparse-style options and ini-style config values,
    called once at the beginning of a test run.

    [...]
    """
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Configuring fixtures with command line options

Another look at the `server_ip` fixture from the fixtures chapter:

```
# fixtures/cli_opt/conftest.py
def pytest_addoption(parser: pytest.Parser) -> None:
    parser.addoption("--server-ip", type=str)

@pytest.fixture
def server_ip(request: pytest.FixtureRequest) -> str:
    return request.config.option.server_ip
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Reporting hooks

```python
def pytest_report_header(
    config: pytest.Config,
    start_path: pathlib.Path,
) -> str | list[str]:
    """Return a string or list of strings to be displayed
    as header info for terminal reporting."""

def pytest_terminal_summary(
    terminalreporter,   # currently undocumented...
    exitstatus: pytest.ExitCode,
    config: pytest.Config,
) -> None:
   """Add a section to terminal summary reporting."""
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Adding to header and summary

```python
# hooks/reporting/conftest.py

def pytest_report_header() -> list[str]:
    return ["extrainfo: line 1"]

def pytest_terminal_summary(terminalreporter) -> None:
    if terminalreporter.verbosity >= 1:
        terminalreporter.section("my special section")
        terminalreporter.line("report something here")
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Adding to header and summary

```
$ pytest
======================== test session starts ========================
platform linux -- Python [...], pytest-[...], pluggy-[...]
extrainfo: line 1
[...]
======================= no tests ran in 0.00s =======================
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Adding to header and summary

```
$ pytest
========================= test session starts =========================
platform linux -- Python [...], pytest-[...], pluggy-[...]
extrainfo: line 1
[...]
======================== no tests ran in 0.00s ========================
```

```
$ pytest -v
========================= test session starts =========================
platform linux -- Python [...], pytest-[...], pluggy-[...]
extrainfo: line 1
[...]
========================= my special section ==========================
report something here
======================== no tests ran in 0.00s ========================
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Anatomy of a pytest run

```
main()
├── Import and register built-in plugins:
│   pytest_plugin_registered
├── pytest_addoption
├── pytest_cmdline_parse
└── pytest_cmdline_main
    ├── pytest_configure
    ├── pytest_sessionstart
    │   └── pytest_report_header
    ├── pytest_collection
    │   └── ...
    ├── pytest_runtestloop
    │   └── ...
    ├── pytest_sessionfinish
    │   └── pytest_terminal_summary
    └── pytest_unconfigure
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Anatomy of a pytest run

Collecting files

```
pytest_collection
├── pytest_collectstart
├── pytest_make_collect_report
│   ├── pytest_ignore_collect
│   ├── pytest_collect_file
│   │   └── pytest_pycollect_makemodule
│   ├── pytest_ignore_collect
│   ├── pytest_collect_file
│   │   └── pytest_pycollect_makemodule
│   └── (repeat)
├── pytest_itemcollected
├── (repeat)
├── pytest_collectreport
└── ...
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Anatomy of a pytest run

Collecting tests

```
pytest_collection
├── ...
├── pytest_collectstart
├── pytest_make_collect_report
│   ├── pytest_pycollect_makeitem
│   ├── pytest_pycollect_makeitem
│   │   └── pytest_generate_tests
│   │       └── pytest_make_parametrize_id
│   └── (repeat)
├── pytest_itemcollected
├── (repeat)
├── pytest_collectreport
├── pytest_collection_modifyitems
└── pytest_collection_finish
    └── pytest_report_collectionfinish
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Anatomy of a pytest run

Collecting tests

- pytest_collection
  - ...
  - pytest_collectstart
  - pytest_make_collect_report
    - pytest_pycollect_makeitem
    - pytest_pycollect_makeitem
      - pytest_generate_tests
        - pytest_make_parametrize_id
    - (repeat)
  - pytest_itemcollected
  - (repeat)
  - pytest_collectreport
  - pytest_collection_modifyitems
  - pytest_collection_finish
    - pytest_report_collectionfinish

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

The pytest_collection_modifyitems hook

```python
def pytest_collection_modifyitems(
    session: pytest.Session,
    config: pytest.Config,
    items: list[pytest.Item]
) -> None:
    """Called after collection has been performed.
    May filter or re-order the items in-place.
    """
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

The pytest_collection_modifyitems hook

```
def pytest_collection_modifyitems(
    session: pytest.Session,
    config: pytest.Config,
    items: list[pytest.Item]
) -> None:
    """Called after collection has been performed.
    May filter or re-order the items in-place.
    """
```

We can use the hook to e.g. add a marker to all tests using a certain fixture:

```
def pytest_collection_modifyitems(items: list[pytest.Item]) -> None:
    for item in items:
        if "qapp" in getattr(item, "fixturenames", []):
            item.add_marker("gui")
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Anatomy of a pytest run

```
main()
├── Import and register built-in plugins:
│   pytest_plugin_registered
├── pytest_addoption
├── pytest_cmdline_parse
└── pytest_cmdline_main
    ├── pytest_configure
    ├── pytest_sessionstart
    │   └── pytest_report_header
    ├── pytest_collection
    │   └── ...
    ├── pytest_runtestloop
    │   └── ...
    ├── pytest_sessionfinish
    │   └── pytest_terminal_summary
    └── pytest_unconfigure
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Anatomy of a pytest run

Running tests

```
pytest_runtestloop
├── pytest_runtest_protocol
│   ├── pytest_runtest_logstart
│   ├── pytest_runtest_setup
│   ├── pytest_runtest_makereport
│   ├── pytest_runtest_logreport
│   │   └── pytest_runtest_teststatus
│   ├── pytest_runtest_call
│   │   └── pytest_pyfunc_call
│   ├── ...  (makereport, logreport)
│   ├── pytest_runtest_teardown
│   ├── ...  (makereport, logreport)
│   └── pytest_runtest_logfinish
└── (repeat)
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

pytest collection objects

- Tree of `Collector` (has children) and `Item` (leaf node; runnable test) instances. Both are `Node` subclasses.
- Collectors' `collect()` returns list of child `Collector` and `Item` instances.
- Items' `runtest()` runs a test.
- Test collection starts from directories or files.

Inspect the test collection tree:
`pytest --collect-only`

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

pytest collection objects

Attributes on Items and Collectors:

- **parent** A reference to the collector that produced us
- **name** A name that is unique within the scope of our parent
- **config** Global configuration object

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Custom domain-specific language (DSL) and collector

```yaml
# hooks/yaml/test_calc.yml
- name: adding-stack
  inputs: ["10", "20"]
  stack: [10, 20]

- name: simple-addition
  inputs: ["10", "5", "+"]
  stack: [15]
```

pytest lets us build a plugin for:

- collecting tests by parsing `.yml` files
- running tests according to plugin-specified code
- reporting in a plugin-specified manner

Such plugin-specific tests can run alongside "normal" tests

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Domain specific testing languages (DSLs)

Domain Specific Languages for testing allow us to:

- Choose the right level of abstraction for test specification
- Write tests without Python knowledge
- Change test implementation without modifying test specs

But beware:

- Don't start by inventing new data format/syntax
- Evolve DSLs by actual needs

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Example: using YAML as language syntax

YAML ("YAML Ain't Markup Language"):

- Format for list/dict/string/number data structures
- Very human readable, yet machine parseable
- Used in many popular projects

Syntax quickstart:
docs.ansible.com → Ansible Documentation → YAML Syntax

https://docs.ansible.com/ansible/latest/reference_appendices/YAMLSyntax.html

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

YAML example file

Here is a list containing one dict in YAML:

```yaml
- name: simple-addition
  inputs: ["10", "5", "+"]
  stack: [15]
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

YAML example file

Here is a list containing one dict in YAML:

```yaml
- name: simple-addition
  inputs: ["10", "5", "+"]
  stack: [15]
```

In Python:

```python
[
    {
        "name": "simple-addition",
        "inputs": ["10", "5", "+"],
        "stack": [15]
    }
]
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

YAML example file

Here is a list containing one dict in YAML:

```yaml
- name: simple-addition
  inputs: ["10", "5", "+"]
  stack: [15]
```

In Python:

```python
[
    {
        "name": "simple-addition",
        "inputs": ["10", "5", "+"],
        "stack": [15]
    }
]
```

Parsing with the PyYAML package:

```python
import yaml
with open("test_calc.yml") as f:
    items = yaml.safe_load(f.read())
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

YAML pytest implementation

The plugin consists of four parts:

- The `pytest_collect_file` hook creates `YamlFile` (`Collector`) nodes for `.yml` files
- `YamlFile` parses out a list of test items (`YamlItem`)
- `YamlItem.runtest()` performs the test, possibly raising exceptions
- `YamlItem.repr_failure()` performs reporting in case of failures

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

YAML test file example

Our test file:

```yaml
# hooks/yaml/test_calc.yml
- name: adding-stack
  inputs: ["10", "20"]
  stack: [10, 20]

- name: simple-addition
  inputs: ["10", "5", "+"]
  stack: [15]
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

YAML test file example

Our test file:

```yaml
# hooks/yaml/test_calc.yml
- name: adding-stack
  inputs: ["10", "20"]
  stack: [10, 20]

- name: simple-addition
  inputs: ["10", "5", "+"]
  stack: [15]
```

Plugin code steps:

- Create `RPNCalculator`
- Pass inputs from `inputs:` to it
- Make sure the stack is equal to `stack:`

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

YAML test file example

Our test file:

```yaml
# hooks/yaml/test_calc.yml
- name: adding-stack
  inputs: ["10", "20"]
  stack: [10, 20]

- name: simple-addition
  inputs: ["10", "5", "+"]
  stack: [15]
```

Plugin code steps:

- Create `RPNCalculator`
- Pass inputs from `inputs:` to it
- Make sure the stack is equal to `stack:`

Exercise:

- Install PyYAML via `pip install pyyaml`
- In `hooks/yaml/`, run `pytest test_calc.yml`, try `--collect-only`
- Add another (failing?) test to `test_calc.yml`, rerun `pytest test_calc.yml`
- Take a first look at the code in `conftest.py`, it'll be explained on the next slide

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

YAML plugin code

```python
def pytest_collect_file(parent, file_path):
    if file_path.suffix == ".yml" and file_path.name.startswith("test"):
        return YamlFile.from_parent(parent, path=file_path)


class YamlFile(pytest.File):
    def collect(self):
        with self.path.open("r") as f:
            raw = yaml.safe_load(f)

        for spec in raw:
            name = spec["name"]
            yield YamlItem.from_parent(self, name=name, spec=spec)
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

YAML plugin code

```
def pytest_collect_file(parent, file_path):
    if file_path.suffix == ".yml" and file_path.name.startswith("test"):
        return YamlFile.from_parent(parent, path=file_path)


class YamlFile(pytest.File):
    def collect(self):
        with self.path.open("r") as f:
            raw = yaml.safe_load(f)

        for spec in raw:
            name = spec["name"]
            yield YamlItem.from_parent(self, name=name, spec=spec)
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

YAML plugin code

```
def pytest_collect_file(parent, file_path):
    if file_path.suffix == ".yml" and file_path.name.startswith("test"):
        return YamlFile.from_parent(parent, path=file_path)


class YamlFile(pytest.File):
    def collect(self):
        with self.path.open("r") as f:
            raw = yaml.safe_load(f)

        for spec in raw:
            name = spec["name"]
            yield YamlItem.from_parent(self, name=name, spec=spec)
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

YAML plugin code

```
def pytest_collect_file(parent, file_path):
    if file_path.suffix == ".yml" and file_path.name.startswith("test"):
        return YamlFile.from_parent(parent, path=file_path)


class YamlFile(pytest.File):
    def collect(self):
        with self.path.open("r") as f:
            raw = yaml.safe_load(f)

        for spec in raw:
            name = spec["name"]
            yield YamlItem.from_parent(self, name=name, spec=spec)
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

YAML plugin code

```
class YamlItem(pytest.Item):
    def __init__(self, name, parent, spec):
        super().__init__(name, parent)
        self.spec = spec

    def runtest(self):
        """Run the test, raise exceptions on issues."""

    def repr_failure(self, excinfo):
        """Called when self.runtest() raises an exception."""

    def reportinfo(self):
        """Return the path/linenumber/text to show."""
        return self.fspath, 0, f"usecase: {self.name}"
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

YAML plugin code

```
class YamlItem(pytest.Item):
    [...]

    def runtest(self):
        """Run the test, raise exceptions on issues."""
        rpn = RPNCalculator(Config())
        try:
            inputs = self.spec["inputs"]
            stack = self.spec["stack"]
        except KeyError as e:
            raise YamlException(f"Missing key: {e}")

        for inp in inputs:
            rpn.evaluate(inp)

        assert rpn.stack == stack
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

YAML plugin code

```
class YamlItem(pytest.Item):
    [...]

    def runtest(self):
        """Run the test, raise exceptions on issues."""
        rpn = RPNCalculator(Config())
        try:
            inputs = self.spec["inputs"]
            stack = self.spec["stack"]
        except KeyError as e:
            raise YamlException(f"Missing key: {e}")

        for inp in inputs:
            rpn.evaluate(inp)

        assert rpn.stack == stack
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

YAML plugin code

```
class YamlItem(pytest.Item):
    [...]

    def runtest(self):
        """Run the test, raise exceptions on issues."""
        rpn = RPNCalculator(Config())
        try:
            inputs = self.spec["inputs"]
            stack = self.spec["stack"]
        except KeyError as e:
            raise YamlException(f"Missing key: {e}")

        for inp in inputs:
            rpn.evaluate(inp)

        assert rpn.stack == stack
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

YAML plugin code

```
class YamlItem(pytest.Item):
    [...]

    def runtest(self):
        """Run the test, raise exceptions on issues."""
        rpn = RPNCalculator(Config())
        try:
            inputs = self.spec["inputs"]
            stack = self.spec["stack"]
        except KeyError as e:
            raise YamlException(f"Missing key: {e}")

        for inp in inputs:
            rpn.evaluate(inp)

        assert rpn.stack == stack
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

YAML plugin code

```
class YamlException(Exception):
    """Custom exception for error reporting."""

class YamlItem(pytest.Item):
    [...]
    def runtest(self):
        [...]
        except KeyError as e:
            raise YamlException(f"Missing key: {e}")
        [...]

    def repr_failure(self, excinfo):
        """Called when self.runtest() raises an exception."""
        if isinstance(excinfo.value, YamlException):
            return f"Invalid YAML: {excinfo.value}"
        return super().repr_failure(excinfo)
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Making a plugin installable

```python
# hooks/yaml/setup.py
from setuptools import setup

if __name__ == "__main__":
    setup(
        name='pytest-yamlcalc', version='0.1',
        py_modules=['pytest_yamlcalc'],
        entry_points={'pytest11': ['yamlcalc = pytest_yamlcalc']},
        install_requires=['pytest>=7.0.0', 'PyYAML'],
        description='yaml-based rpn calculator tests',
        author='yourname',
        author_email='you@example.com',
        url='http://www.example.com/url/',
        license="MIT",
    )
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Cookiecutter makes things easier

- `pip install cookiecutter`
- `cookiecutter https://github.com/pytest-dev/cookiecutter-pytest-plugin`

```
pytest-yamlcalc/
├── .gitignore
├── appveyor.yml
├── docs
│   ├── conf.py
│   ├── index.rst
│   └── ...
├── LICENSE
├── MANIFEST.in
├── pytest_yamlcalc.py
├── README.rst
├── setup.py
├── tests
│   ├── conftest.py
│   └── test_yamlcalc.py
└── tox.ini
```

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

DSL testing plugins

Lessons learned:

- pytest is flexible about how a test looks – it doesn't need to be written in Python!
- DSL tests can run in addition to Python ones
- DSL can allow specifying tests without Python knowledge
- Can enhance plugin implementation (e.g. reporting) without modifying test specification
- YAML is a good basis for writing a DSL

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Book recommendation

Brian Okken: Python Testing with pytest, Second Edition (The Pragmatic Bookshelf)

- ISBN 978-1680508604
- https://pragprog.com/titles/bopytest2/
- Discount code: **EuroPythonPrague2023**
  35% discount on PDF + epub + mobi until August 10
- Full disclosure: I'm technical reviewer (but don't earn any money from it)

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Next trainings

- **September 5th, 2023:**
  CH Open Workshoptage (workshoptage.ch):
  pytest: Professionelles Testen (nicht nur) für Python
  Rapperswil (Switzerland)
- **March 5th to 7th, 2024:**
  Python Academy (python-academy.com):
  Professional Testing with Python
  Leipzig (Germany) and remote
- **Custom training / coaching** (remote or on-site):
  - Python
  - pytest
  - Best Practices (packaging, linting, etc.)
  - GUI programming with Qt, Git, ...

BRUHIN
SOFTWARE

florian@bruhin.software, https://bruhin.software/

© Florian Bruhin / Bruhin Software, all rights reserved. For personal use by Europython 2023 attendees only!

Feedback and questions

Florian Bruhin
florian@bruhin.software
https://bruhin.software/
@the_compiler on Twitter

BRUHIN
SOFTWARE

Copyright 2015 – 2023 Florian Bruhin, all rights reserved.
Reuse or redistribution is prohibited without prior written permission.

Originally based on materials copyright 2013 – 2015 by Holger Krekel, used with permission.