Professionelles Testen für Python mit pytest

Florian Bruhin / “The Compiler”

pytest

BRUHIN
SOFTWARE

15. April 2021

Before we get started...

About you!

I like writing tests

I usually don't write tests

I regularly use pytest

I regularly use unittest.py

Before we get started...

About me

❤️ Like tests 😲 Don't write tests 👏 pytest 👍 unittest.py

2011 Started using Python
2013 Started developing qutebrowser, writing tests
2015 Switched to pytest, ended up as a maintainer

40% employed (OST: Eastern Switzerland University of Applied Sciences),
60% open-source and freelancing (Bruhin Software)

Before we get started...

About me

❤️ Like tests 😲 Don't write tests 👏 pytest 👍 unittest.py

2011 Started using Python
2013 Started developing qutebrowser, writing tests
2015 Switched to pytest, ended up as a maintainer

40% employed (OST: Eastern Switzerland University of Applied Sciences),
60% open-source and freelancing (Bruhin Software)

Before we get started...

About me

❤️ Like tests 😲 Don't write tests 👏 pytest 👍 unittest.py

2011 Started using Python
2013 Started developing qutebrowser, writing tests
2015 Switched to pytest, ended up as a maintainer

40% employed (OST: Eastern Switzerland University of Applied Sciences),
60% open-source and freelancing (Bruhin Software)

Before we get started...

About me

❤️ Like tests 😲 Don't write tests 👏 pytest 👍 unittest.py

2011 Started using Python
2013 Started developing qutebrowser, writing tests
2015 Switched to pytest, ended up as a maintainer

40% employed (OST: Eastern Switzerland University of Applied Sciences),
60% open-source and freelancing (Bruhin Software)

Before we get started...

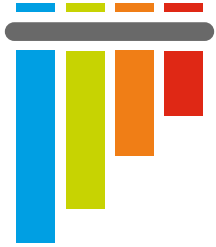
About me

❤️ Like tests 😲 Don't write tests 👏 pytest 👍 unittest.py

2011 Started using Python
2013 Started developing qutebrowser, writing tests
2015 Switched to pytest, ended up as a maintainer

BRUHIN
SOFTWARE

40% employed (OST: Eastern Switzerland University of Applied Sciences),
60% open-source and freelancing (Bruhin Software)



pytest

Why pytest?

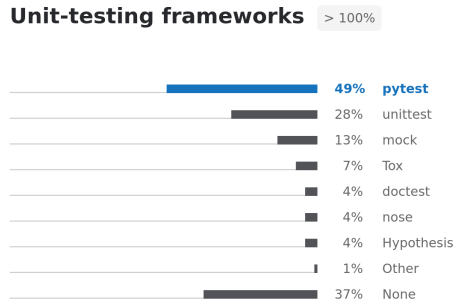
Features

- Automatic test discovery,
  no-boilerplate test code
  (boilerplate: repeated code without any “real” use)
- Useful information when a test fails
- Test parametrization
- Modular setup/teardown via fixtures
- Customizable: Many options, hundreds of useful plugins

Why pytest?

Popularity

- Automatic test discovery, no-boilerplate test code
  (boilerplate: repeated code without any “real” use)
- Useful information when a test fails
- Test parametrization
- Modular setup/teardown via fixtures
- Customizable: Many options, hundreds of useful plugins



(JetBrains Python Developers Survey 2020)

No boilerplate

Example code

```
def add_two(val):
    return val + 2
```

No boilerplate

Test with unittest.py

```
import unittest

class AddTwoTests(unittest.TestCase):

    def testAddTwo(self):
        self.assertEqual(add_two(2), 4)

if __name__ == '__main__':
    unittest.main()
```

No boilerplate

Test with pytest

```
import unittest

class AddTwoTests(unittest.TestCase):

    def testAddTwo(self):
        self.assertEqual(add_two(2), 4)

if __name__ == '__main__':
    unittest.main()
```

```
def test_add_two():
    assert add_two(2) == 4
```

No boilerplate

Assert introspection

```
                          # with unittest.py:
assert x                  # assertTrue(x)
assert x == 1             # assertEqual(x, 1)
assert x != 2             # assertNotEqual(x, 2)
assert not x              # assertFalse(x)
assert x < 3 or y > 5     # ?
```

No boilerplate

Failing tests

```
_____________________________ test_failure _____________________________

    def test_failure():
        a = "Hello World!"
        b = "Hello, World!"
>       assert a == b
E       AssertionError: assert 'Hello World!' == 'Hello, World!'
E         - Hello, World!
E         ?      -
E         + Hello World!

test_output.py:4: AssertionError
======================== short test summary info ========================
FAILED test_output.py::test_failure - AssertionError: assert 'Hello...
========================= 1 failed in 0.05s =========================
```

No boilerplate

Other output examples

```
    def test_eq_list():
>       assert [0, 1, 2] == [0, 1, 3]
E       assert [0, 1, 2] == [0, 1, 3]
E         At index 2 diff: 2 != 3
E         Use -v to get the full diff
```

No boilerplate

Other output examples

```
    def test_eq_list():
>       assert [0, 1, 2] == [0, 1, 3]
E       assert [0, 1, 2] == [0, 1, 3]
E         At index 2 diff: 2 != 3
E         Use -v to get the full diff

    def test_not_in_text():
        text = "single foo line"
>       assert "foo" not in text
E       AssertionError: assert 'foo' not in 'single foo line'
E         'foo' is contained here:
E           single foo line
E         ?        +++
```

Markers

Markers

Skipping

```
@pytest.mark.skipif(
    sys.platform != 'win32',
    reason="Only runs on Windows",
)
def test_windows_features():
    assert False
```

Markers

Skipping

```python
@pytest.mark.skipif(
    sys.platform != 'win32',
    reason="Only runs on Windows",
)
def test_windows_features():
    assert False
```

```
======================== test session starts ========================
collecting ... collected 1 item

test_skipping.py::test_windows_features SKIPPED (Only runs...) [100%]

========================= 1 skipped in 0.00s =========================
```

Markers

Expected to fail

```
@pytest.mark.xfail(reason="See JIRA-2342")
def test_broken_api():
    assert False
```

Markers

Expected to fail

```
@pytest.mark.xfail(reason="See JIRA-2342")
def test_broken_api():
    assert False
```

```
======================== test session starts ========================
collecting ... collected 1 item

test_xfail.py::test_broken_api XFAIL (See JIRA-2342)          [100%]

======================== 1 xfailed in 0.01s =========================
```

Markers

Unexpected pass

```python
@pytest.mark.xfail(reason="See JIRA-2342")
def test_broken_api():
    pass
```

```
======================== test session starts ========================
collecting ... collected 1 item

test_xpass.py::test_broken_api XPASS (See JIRA-2342)           [100%]

========================= 1 xpassed in 0.00s =========================
```

Markers

Custom markers

```
@pytest.mark.slow
def test_slow():
    time.sleep(2)

def test_fast_1():
    pass

def test_fast_2():
    pass
```

Markers

Custom markers

```
@pytest.mark.slow
def test_slow():
    time.sleep(2)

def test_fast_1():
    pass

def test_fast_2():
    pass
```

```
[pytest]
markers =
  slow: Tests which take a while to run
  ...
```

Markers

Custom markers

```
@pytest.mark.slow
def test_slow():
    time.sleep(2)

def test_fast_1():
    pass

def test_fast_2():
    pass
```

```
[pytest]
markers =
  slow: Tests which take a while to run
  ...
```

```
$ pytest -m "not slow"
=============== test session starts ===============
collected 3 items / 1 deselected / 2 selected

test_custom_marker.py::test_fast_1 PASSED
test_custom_marker.py::test_fast_2 PASSED

========= 2 passed, 1 deselected in 0.00s =========
```

Markers

Custom markers with data

```
@pytest.mark.use_config("production.yml")
def test_prod_config():
    ...
```

Parametrizing

Parametrizing

with unittest.py

```
import unittest

class AddTests(unittest.TestCase):

    def testMinusOne(self):
        self.assertEqual(add_two(-1), 1)

    def testZero(self):
        self.assertEqual(add_two(0), 2)

    def testTwo(self):
        self.assertEqual(add_two(2), 4)
```

Parametrizing

with pytest

```python
import unittest

class AddTests(unittest.TestCase):

    def testMinusOne(self):
        self.assertEqual(add_two(-1), 1)

    def testZero(self):
        self.assertEqual(add_two(0), 2)

    def testTwo(self):
        self.assertEqual(add_two(2), 4)
```

```python
import pytest

@pytest.mark.parametrize(
    'inp, out', [
        (-1, 1),
        (0, 2),
        (2, 4),
    ]
)
def test_add_two(inp, out):
    assert add_two(inp) == out
```

Parametrizing

Result

```
@pytest.mark.parametrize('inp, out', [(-1, 1), (0, 2), (2, 4)])
def test_add_two(inp, out):
    assert add_two(inp) == out
```

```
======================== test session starts ========================
[...]

test_param.py::test_add_two[-1-1] PASSED                        [ 33%]
test_param.py::test_add_two[0-2] PASSED                         [ 66%]
test_param.py::test_add_two[2-4] PASSED                         [100%]

========================= 3 passed in 0.01s =========================
```

Fixtures

Fixtures

Basic example

```python
import pytest

@pytest.fixture
def answer():
    return 42

def test_answer(answer):
    assert answer == 42
```

Fixtures

Basic example

```python
import pytest

@pytest.fixture
def answer():
    return 42

def test_answer(answer):
    assert answer == 42
```

Fixtures

Fixtures using fixtures

```
import pytest

@pytest.fixture
def half():
    return 21

@pytest.fixture
def answer( half ):
    return half * 2

def test_answer(answer):
    assert answer == 42
```

Fixtures

Setup and teardown

```
@pytest.fixture
def database():
    db = Database()
    db.connect()
    yield db
    db.rollback()

def test_database(database):
    ...
```

Fixtures

Other features

- Caching fixture values: `@pytest.fixture(scope="module")`
- Using fixtures implicitly: `@pytest.fixture(autouse=True)`
- Running tests with differently configured resources:
  `@pytest.fixture(params=[Postgres(), MariaDB()])`

Builtin fixtures

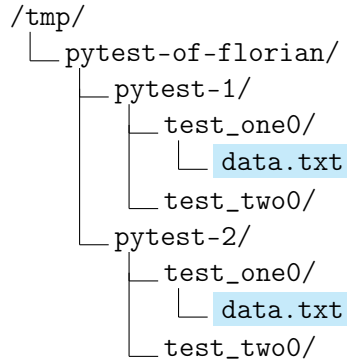
tmp_path

```python
def test_one(tmp_path):
    input_file = tmp_path / "data.txt"
    input_file.write_text("Hello World")

def test_two(tmp_path):
    assert not list(tmp_path.iterdir())
```

Builtin fixtures

tmp_path

```python
def test_one(tmp_path):
    input_file = tmp_path / "data.txt"
    input_file.write_text("Hello World")

def test_two(tmp_path):
    assert not list(tmp_path.iterdir())
```



Builtin fixtures

monkeypatch

```
def test_debug_mode(monkeypatch):
    monkeypatch.setenv('MYAPP_DEBUG', '1')
    assert 'MYAPP_DEBUG' in os.environ

def test_something_else():
    assert 'MYAPP_DEBUG' not in os.environ
```

Builtin fixtures

monkeypatch

```
def test_debug_mode(monkeypatch):
    monkeypatch.setenv('MYAPP_DEBUG', '1')
    assert 'MYAPP_DEBUG' in os.environ

def test_something_else():
    assert 'MYAPP_DEBUG' not in os.environ

def test_fake_windows(monkeypatch):
    monkeypatch.setattr(sys, 'platform', 'win32')
    ...
```

Builtin fixtures

Capturing

```
def test_output(capsys):
    print("Hello World")
    stdout, stderr = capsys.readouterr()
    assert stdout == "Hello World\n"
```

Plugins

Plugins / Related projects

coverage.py / pytest-cov

```
92  def download_dir():
93      """Get the download directory to use."""
94      directory = config.val.downloads.location.directory
95      remember_dir = config.val.downloads.location.remember
96
97      if remember_dir and last_used_directory is not None:
98          ddir = last_used_directory
99      elif directory is None:
100         ddir = standarddir.download()
101     else:
102         ddir = directory
103
104     try:
105         os.makedirs(ddir, exist_ok=True)
106     except OSError as e:
107         message.error("Failed to create download directory: {}".format(e))
108
109     return ddir
```

Plugins / Related projects

pytest-bdd

```
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
```

Plugins / Related projects

pytest-bdd

```
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
```

```
@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()
```

Plugins / Related projects

Hypothesis

```
@given(text())
def test_decode_inverts_encode(s):
    assert decode(encode(s)) == s
```

Plugins / Related projects

Hypothesis

```
@given(text())
def test_decode_inverts_encode(s):
    assert decode(encode(s)) == s


Falsifying example: test_decode_inverts_encode(s='')

UnboundLocalError: local variable 'character' referenced
before assignment
```

Plugins / Related projects

Plugins, plugins, plugins...

- Property-based testing: hypothesis
- Customized reporting: pytest-html, pytest-sugar, 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)
- ... >800 more:
  `https://docs.pytest.org/en/latest/reference/plugin_list.html`

Plugins / Related projects

Plugins are easy!

```
# conftest.py

def pytest_addoption(parser):
    parser.addoption("-backend", choices=("webkit", "webengine"),
                     default="webkit")

@pytest.fixture
def backend_arg(request):
    return request.config.getoption("-backend")

# test_something.py

def test_something(backend_arg):
    # ...
```

Plugins / Related projects

Plugins are easy!

```
# conftest.py

def pytest_addoption(parser):
    parser.addoption("-backend", choices=("webkit", "webengine"),
                     default="webkit")

@pytest.fixture
def backend_arg(request):
    return request.config.getoption("-backend")

# test_something.py

def test_something(backend_arg):
    # ...
```

Plugins / Related projects

Plugins are easy!

```
# conftest.py

def pytest_addoption(parser):
    parser.addoption("-backend", choices=("webkit", "webengine"),
                     default="webkit")

@pytest.fixture
def backend_arg(request):
    return request.config.getoption("-backend")

# test_something.py

def test_something(backend_arg):
    # ...
```

Plugins / Related projects

Domain-specific languages

```
- name: mp3-compression
  type: check-compression
  codec: mp3
  inputfile: data1.wav
  compression: 10%
```

Plugins / Related projects

Domain-specific languages

```yaml
- name: mp3-compression
  type: check-compression
  codec: mp3
  inputfile: data1.wav
  compression: 10%
```

```html
<!DOCTYPE html>

<!- target: hello.txt ->

<html>
  <head>...</head>
  <body>
    <a href="/data/hello.txt" id="link">
      Follow me!</a>
  </body>
</html>
```

Where to go from there...

Switching to pytest

- pytest can run existing `unittest.py` and `nose` tests!
- If you want to rewrite your tests, `unittest2pytest` can help:

```
class TestExample(unittest.TestCase):

    def testExample(self):
        self.assertEqual(1, 2)
```

```
def testExample(self):
    assert 1 == 2
```

Trainings

- As part of enterPy
  - 22nd April (09:00 – 16:00, fully booked)
  - 20th May (09:00 – 16:00)
- In-house at your company
  - Three-day pytest/tox/devpi deep-dive
  - Various other topics (Python basics, advanced Python, best practices, GUI applications with Qt, ...)
  - Or tailored to your needs!

Contact and resources

Florian Bruhin
florian@bruhin.software
https://bruhin.software/
@the_compiler on Twitter

Are you interested in customized trainings, development or consulting? Let's talk!

BRUHIN
SOFTWARE

https://pytest.org
https://github.com/pytest-dev/pytest
irc.freenode.net → #pylib