Sharing is Caring

pytest Fixture Edition

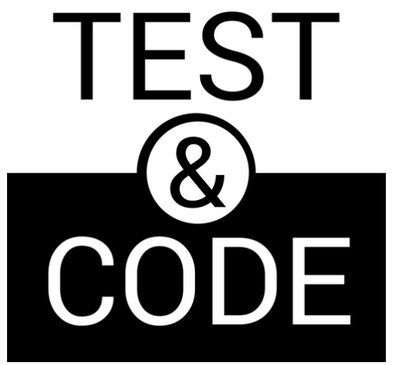
Brian Okken

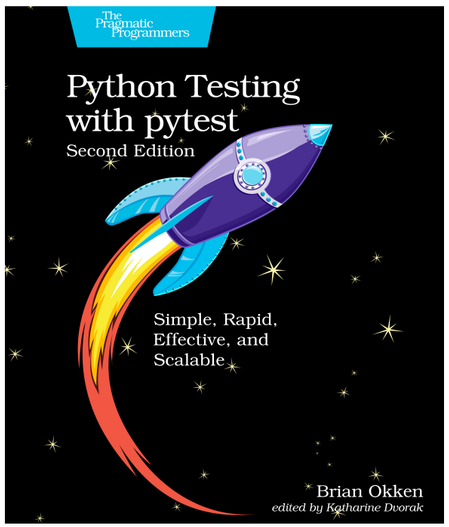
Code examples are here

github.com/okken/pytest_fixture_sharing

Who is Brian Okken

- Podcasts
  - Python Bytes
  - Test and Code
- Book
  - Python Testing with pytest
- Lead Software Engineer





Sharing pytest Fixtures

- pytest is awesome
- fixtures are awesome
- you are awesome
- you will make fixtures so awesome you want to share them

Fixture crash course

typical example

```
import pytest

@pytest.fixture()
def db():
    ... # setup
    yield _db
    ... # teardown
```

typical example

```python
import pytest

@pytest.fixture()
def db():
    ... # setup
    yield _db
    ... # teardown
```

```python
def test_foo(db):
    ...
    result = db.action()
    ...

def test_bar(db):
    ...
    result = db.action()
    ...
```

but frankly

that's too useful for this talk

let's do something silly

like colorful

print statements

```
(sharing) $ pytest test_example.py
============================= test session starts ==============================
collected 2 items

test_example.py
----- test_example.py::test_square: x=7, y=49, expected=49
.
----- test_example.py::test_cube: x=7, y=343, expected=343
.

============================== 2 passed in 0.04s ===============================
```

test_example.py (the fixture)

```
import pytest
import rich

@pytest.fixture()
def blue(request, capsys):
    def _blue(text: str):
        with capsys.disabled():
            test_func = request.node.nodeid
            rich.print(f'\n[blue]----- {test_func}: {text}')
    return _blue
```

test_example.py (the tests)

```
def test_square(blue):
    x = 7
    y = x**2
    expected = x * x
    blue(f'{x=}, {y=}, {expected=}')
    assert y == expected

def test_cube(blue):
    x = 7
    y = pow(x, 3)
    expected = x * x * x
    blue(f'{x=}, {y=}, {expected=}')
    assert y == expected
```

Resulting in

```
(sharing) $ pytest test_example.py
============================= test session starts ==============================
collected 2 items

test_example.py
----- test_example.py::test_square: x=7, y=49, expected=49
.
----- test_example.py::test_cube: x=7, y=343, expected=343
.

========================== 2 passed in 0.04s ===================================
```

Can it be shared?

Right now it's in the test file

test_example.py

```
import pytest
import rich

@pytest.fixture()
def blue(request, capsys):
    def _blue(text: str):
        with capsys.disabled():
            test_func = request.node.nodeid
            rich.print(f'\n[blue]----- {test_func}: {text}')
    return _blue

def test_square(blue):
    x = 7
    ...
```

But we can take it out of there

And put it in a conftest.py file

```
test_directory
├── conftest.py <-- put blue() here
├── test_example.py
└── test_example_two.py
```

conftest.py

```
import pytest
import rich

@pytest.fixture()
def blue(request, capsys):
    def _blue(text: str):
        with capsys.disabled():
            test_func = request.node.nodeid
            rich.print(f'\n[blue]----- {test_func}: {text}')
    return _blue
```

test_example.py

```
def test_square(blue):
    x = 7
    y = x**2
    expected = x * x
    blue(f'{x=}, {y=}, {expected=}')
    assert y == expected

def test_cube(blue):
    x = 7
    y = pow(x, 3)
    expected = x * x * x
    blue(f'{x=}, {y=}, {expected=}')
    assert y == expected
```

test_example_two.py

```
def test_foo(blue):
    blue("this will be blue")
```

In the same directory

```
test_directory
├── conftest.py
├── test_example.py
└── test_example_two.py
```

How about different directories?

As long as conftest.py is in a parent directory

It's still fine

```
test_directory
├── conftest.py
├── one
│   └── test_example.py
└── two
    └── test_example_two.py
```

Can be more conftest.py files

It's still fine

```
test_directory
├── conftest.py
├── one
|   ├── conftest.py
|   └── test_example.py
└── two
    ├── conftest.py
    └── test_example_two.py
```

as long as `blue()` is in a shared parent.

How about sharing between projects?

This is where it gets fun

Before I show you the next couple of slides

- promise you won't run away
- it's actually not as bad as it seems at first

Let's start with

```
blue_starter_project
├── conftest.py
└── test_example.py
```

conftest.py

```
...
@pytest.fixture()
def blue(request, capsys):
    ...
```

test_example.py

```
def test_blue(blue):
    blue('should be blue')
```

And move to

```
blue_starter_project
├── conftest.py
└── test_example.py
```

```
pytest-blue-flit
├── LICENSE.txt
├── README.md
├── examples
│   └── test_example.py
├── pyproject.toml
├── src
│   └── pytest_blue
│       ├── __init__.py
│       └── plugin.py
└── tox.ini
```

Trust me

it's not that bad

Fixture moves from conftest.py to plugin.py

```
blue_starter_project
├── conftest.py
└── test_example.py
```

```
pytest-blue-flit
├── LICENSE.txt
├── README.md
├── examples
│   └── test_example.py
├── pyproject.toml
├── src
│   └── pytest_blue
│       ├── __init__.py
│       └── plugin.py
└── tox.ini
```

test_example.py moves to examples

```
blue_starter_project
├── conftest.py
└── test_example.py
```

```
pytest-blue-flit
├── LICENSE.txt
├── README.md
├── examples
│   └── test_example.py
├── pyproject.toml
├── src
│   └── pytest_blue
│       ├── __init__.py
│       └── plugin.py
└── tox.ini
```

What's left

```
pytest-blue-flit
├── LICENSE.txt
├── README.md
├── examples
│   └── test_example.py
├── pyproject.toml
├── src
│   └── pytest_blue
│       ├── __init__.py
│       └── plugin.py
└── tox.ini
```

LICENSE

```
pytest-blue-flit
├── LICENSE.txt
├── README.md
├── examples
│   └── test_example.py
├── pyproject.toml
├── src
│   └── pytest_blue
│       ├── __init__.py
│       └── plugin.py
└── tox.ini
```

Whatever you want

LICENSE.txt

```
The MIT License (MIT)

Copyright (c) 2020, Brian Okken

Permission is hereby granted, free of charge, ...
```

README

```
pytest-blue-flit
├── LICENSE.txt
├── README.md
├── examples
│   └── test_example.py
├── pyproject.toml
├── src
│   └── pytest_blue
│       ├── __init__.py
│       └── plugin.py
└── tox.ini
```

Whatever you want

README.md

```
# pytest-blue

A pytest plugin that has
one fixture: `blue`

This fixture can ...

Here's an example: ...
```

I like to include

- what it does
- how to install
- an example

pyproject.toml

```
pytest-blue-flit
├── LICENSE.txt
├── README.md
├── examples
│   └── test_example.py
├── pyproject.toml
├── src
│   └── pytest_blue
│       ├── __init__.py
│       └── plugin.py
└── tox.ini
```

Let's come back to that

__init__.py

```
pytest-blue-flit
├── LICENSE.txt
├── README.md
├── examples
│   └── test_example.py
├── pyproject.toml
├── src
│   └── pytest_blue
│       ├── __init__.py
│       └── plugin.py
└── tox.ini
```

__init__.py

```
pytest-blue-flit
├── LICENSE.txt
├── README.md
├── examples
│   └── test_example.py
├── pyproject.toml
├── src
│   └── pytest_blue
│       ├── __init__.py
│       └── plugin.py
└── tox.ini
```

```
__version__ = "0.0.1"
```

pyproject.toml

```
pytest-blue-flit
├── LICENSE.txt
├── README.md
├── examples
│   └── test_example.py
├── pyproject.toml
├── src
│   └── pytest_blue
│       ├── __init__.py
│       └── plugin.py
└── tox.ini
```

the full pyproject.toml

```toml
[project]
name = "pytest-blue"
readme = "README.md"
license = {file = "LICENSE.txt"}
requires-python = ">3.7"
description = "pytest plugin, adds fixture `blue` for debug printing"
dynamic = ["version"]
dependencies = ["pytest", "rich"]
authors = [{name = "Brian Okken"}]
classifiers = ["Framework :: Pytest"]

[project.urls]
Home = "https://github.com/okken/pytest-blue"

[project.entry-points.pytest11]
pytest_blue = "pytest_blue.plugin"

[build-system]
requires = ["flit_core >=3.2,<4"]
build-backend = "flit_core.buildapi"

[tool.flit.module]
name = "pytest_blue"
```

the full pyproject.toml

```toml
[project]
name = "pytest-blue"
readme = "README.md"
license = {file = "LICENSE.txt"}
requires-python = ">3.7"
description = "pytest plugin, adds fixture `blue` for debug printing"
dynamic = ["version"]
dependencies = ["pytest", "rich"]
authors = [{name = "Brian Okken"}]
classifiers = ["Framework :: Pytest"]

[project.urls]
Home = "https://github.com/okken/pytest-blue"

[project.entry-points.pytest11]
pytest_blue = "pytest_blue.plugin"

[build-system]
requires = ["flit_core >=3.2,<4"]
build-backend = "flit_core.buildapi"

[tool.flit.module]
name = "pytest_blue"
```

Project metadata

```
[project]
name = "pytest-blue"
readme = "README.md"
license = {file = "LICENSE.txt"}
requires-python = ">3.7"
description = "pytest plugin, adds fixture `blue` for debug printing"
dynamic = ["version"]
dependencies = ["pytest", "rich"]
authors = [{name = "Brian Okken"}]
classifiers = ["Framework :: Pytest"]
```

URL

```
[project.urls]
Home = "https://github.com/okken/pytest-blue"
```

pytest entry point

```
[project.entry-points.pytest11]
pytest_blue = "pytest_blue.plugin"
```

format: "name_of_plugin = project.module"

```
[project.entry-points.pytest11]
pytest_blue = "pytest_blue.plugin"
```

```
pytest-blue-flit
├── ...
├── src
│   └── pytest_blue
│       ├── __init__.py
│       └── plugin.py
└── tox.ini
```

Build system

```
[build-system]
requires = ["flit_core >=3.2,<4"]
build-backend = "flit_core.buildapi"
```

Flit module/package/directory

```
[tool.flit.module]
name = "pytest_blue"
```

Flit module/package/directory

```
[tool.flit.module]
name = "pytest_blue"
```

```
pytest-blue-flit
├── ...
├── src
│   └── pytest_blue
│       ├── __init__.py
│       └── plugin.py
└── tox.ini
```

project name

```
[project]
name = "pytest-blue"
```

directory name

```
[tool.flit.module]
name = "pytest_blue"
```

Alternatives to flit

- namely hatch and setuptools
- don't have the dash/underscore problem
- but do need help with the version string

hatch

```
[build-system]
requires = ["hatchling"]
build-backend = "hatchling.build"

[tool.hatch.version]
path = "src/pytest_blue/__init__.py"
```

setuptools

```
[build-system]
requires = ["setuptools>=61.0"]
build-backend = "setuptools.build_meta"

[tool.setuptools.dynamic]
version = {attr = "pytest_blue.__version__"}
```

Flit

```toml
[build-system]
requires = ["flit_core >=3.2,<4"]
build-backend = "flit_core.buildapi"

[tool.flit.module]
name = "pytest_blue"
```

That wasn't too bad, was it?

```
[project]
name = "pytest-blue"
readme = "README.md"
license = {file = "LICENSE.txt"}
requires-python = ">3.7"
description = "pytest plugin, adds fixture `blue` for debug printing"
dynamic = ["version"]
dependencies = ["pytest", "rich"]
authors = [{name = "Brian Okken"}]
classifiers = ["Framework :: Pytest"]

[project.urls]
Home = "https://github.com/okken/pytest-blue"

[project.entry-points.pytest11]
pytest_blue = "pytest_blue.plugin"

[build-system]
requires = ["flit_core >=3.2,<4"]
build-backend = "flit_core.buildapi"

[tool.flit.module]
name = "pytest_blue"
```

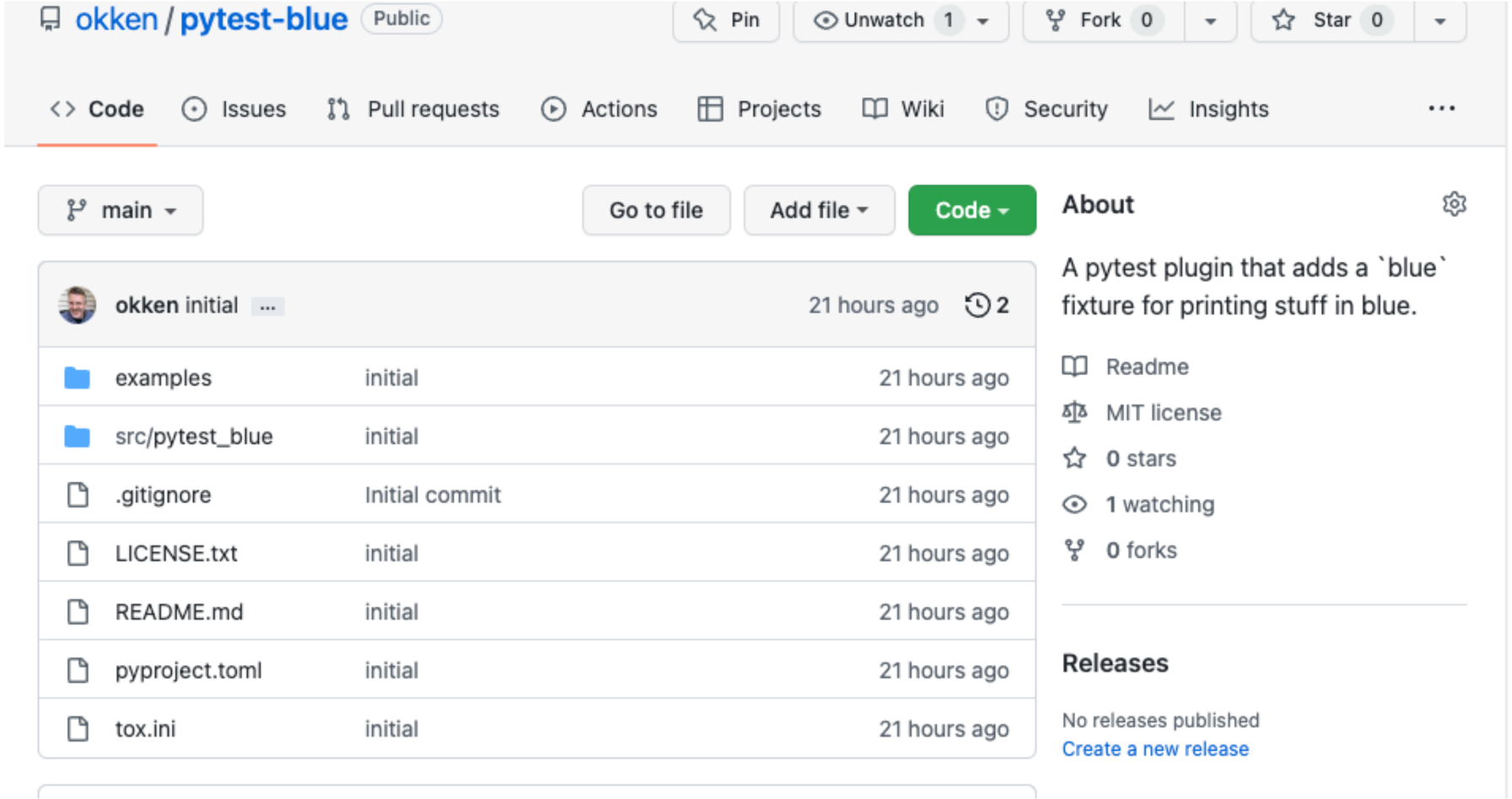
Guess what?

we can share our code now

Seriously

We could just push to a git repo now

Push to github (or other git repo)



Your fixture plugin is now installable

```
$ pip install git+https://github.com/okken/pytest-blue.git
```

or in a `requirements.txt` file

requirements.txt:

```
...
pytest
git+https://github.com/okken/pytest-blue.git
tox
...
```

```
$ pip install -r requirements.txt
```

`pip install` will build the wheel

```
$ pip install git+https://github.com/okken/pytest-blue.git
Collecting ...
  Cloning ...
  Running command git clone ...
  Resolved ...
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
...
Building wheels for collected packages: pytest-blue
  Building wheel for pytest-blue (pyproject.toml) ... done
  Created wheel for pytest-blue: ...
  ...
Successfully built pytest-blue
Installing collected packages: ...
Successfully installed ...
```

IF everything works fine

so maybe

we should do a bit of testing

before we push

enter tox

```
pytest-blue-flit
├── LICENSE.txt
├── README.md
├── examples
│   └── test_example.py
├── pyproject.toml
├── src
│   └── pytest_blue
│       ├── __init__.py
│       └── plugin.py
└── tox.ini
```

You promised

you wouldn't run away

tox.ini

```
pytest-blue-flit
├── LICENSE.txt
├── README.md
├── examples
│   └── test_example.py
├── pyproject.toml
├── src
│   └── pytest_blue
│       ├── __init__.py
│       └── plugin.py
└── tox.ini
```

tox.ini

```
[tox]
envlist = py37, py38, py39, py310, py311,
skip_missing_interpreters = true
isolated_build = True

[testenv]
commands = pytest
description = Run pytest

[pytest]
testpaths = examples
```

Running tox

For example

```
$ pip install tox
$ tox -e py310
```

or all environments

```
$ tox
```

- creates a new empty virtual environment
- builds a wheel
- installs all dependencies
- installs your code
- runs the tests

tox example

```
(sharing) $ tox -q -e py310
============================= test session starts ==============================
collected 1 item

examples/test_example.py
----- examples/test_example.py::test_blue: should be blue
.

============================== 1 passed in 0.04s ===============================
_____________________________________ summary ______________________________________
  py310: commands succeeded
  congratulations :)
(sharing) $
```

ahhh. confidence.

- Now we're testing
  - the fixture
  - the plugin
  - the packaging

Did we really test enough?

- We tested that the plugin can be installed
- And that `blue()` can be called.
- But that's really it.

I mean, this is our test??!!

```
def test_blue(blue):
    blue('should be blue')
```

- Is that enough?
- Should we check that the output happens?
- Or that the text is blue?

Maybe, but….

This is the tox output

```
(sharing) $ tox -q -e py310
============================= test session starts ==============================
collected 1 item

examples/test_example.py
----- examples/test_example.py::test_blue: should be blue
.

============================== 1 passed in 0.04s ===============================
_____________________________________ summary ______________________________________
  py310: commands succeeded
  congratulations :)
(sharing) $
```

And we're one of the customers

Won't we know when it stops working?

It's a judgement call

- There are more tests we can do.
- See docs on `pytester`, especially.
- Or there's this book I know....

But I think we're good for now

Is it ready for PyPI?

Actually, yes.

pytest-blue 0.0.1

```
pip install pytest-blue
```

Latest version

Released: Sep 4, 2022

A pytest plugin that adds a `blue` fixture for printing stuff in blue.

Navigation

- Project description
- Release history
- Download files

Project links

- Home

Statistics

GitHub statistics:

- Stars: 0

Project description

pytest-blue

A pytest plugin that one fixture:

- blue

This fixtures can be used to `print` a string in blue. It will also report the test file and function.

And it works even when pytest is capturing output.

Example **test_example.py**:

```
def test_blue(blue):
    blue('should be blue')
```

We won't go through those details

But essentially it's:

- Register for an account on pypi.org
- Use tool of your choice to push to PyPI:
  - `flit publish`
  - `hatch publish`
  - `twine upload dist/*`

TODIL

(the other day I learned)

From Brett Cannon

PyPI will always reject packages with classifiers beginning with "Private ::".
- pypi.org/classifiers

Preventing PyPI upload

```
classifiers = [
    "Framework :: Pytest",
    "Private :: Do Not Upload"
    ]
```

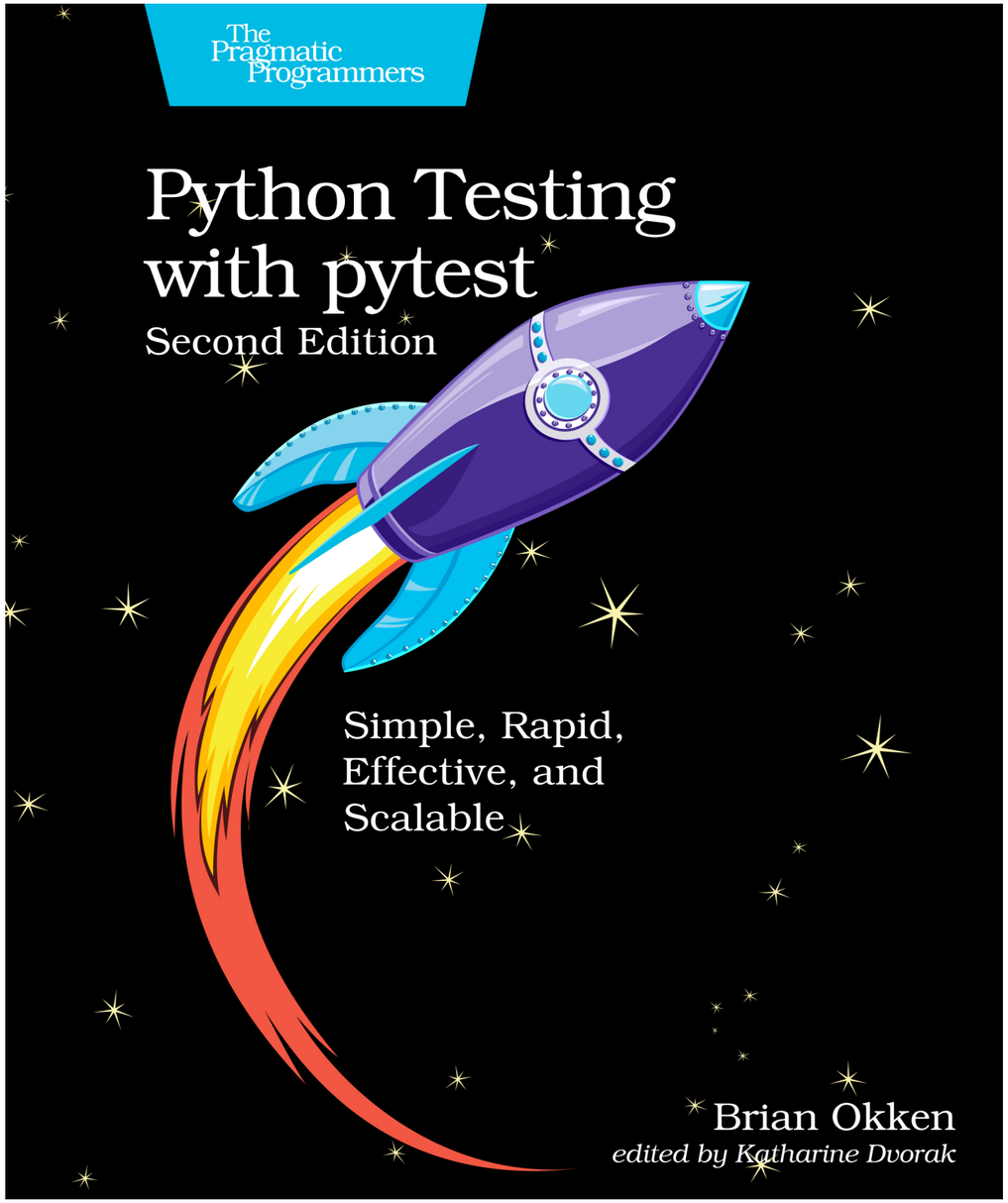
Not too bad, was it?

If you want to know more about

building and testing fixtures and plugins

See:

- Ch 3 for fixtures
- Ch 15 for building plugins
- Ch 11 includes
  - tox
  - GitHub Actions



Stay in touch

- pythontest.com/pytest-book
- pythontest.com/course
- pythontest.com/training
- pythonbytes.fm
- testandcode.com
- pythontest.com
- Twitter @brianokken