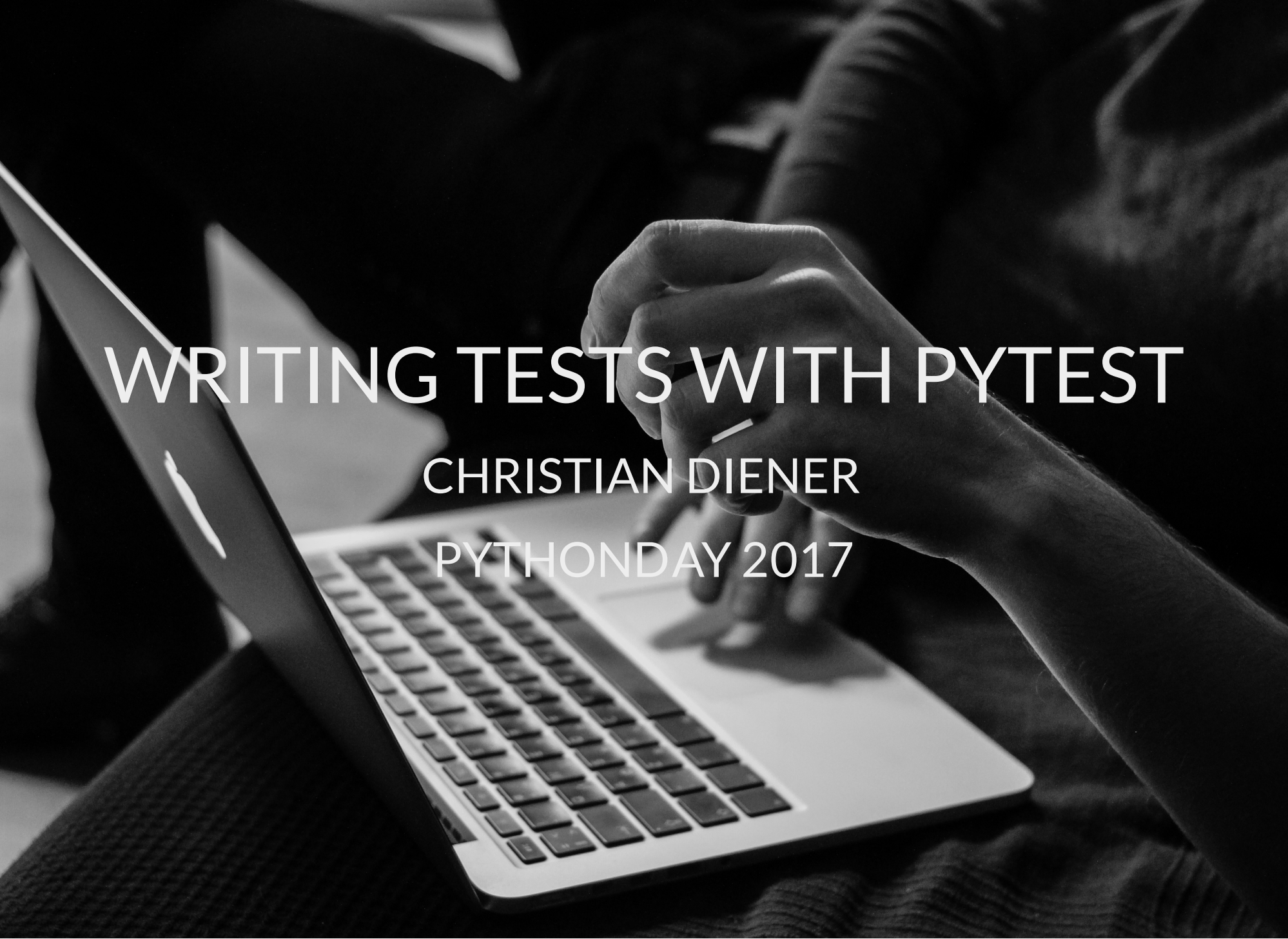


WRITING TESTS WITH PYTEST

CHRISTIAN DIENER

PYTHONDAY 2017

OUR EXAMPLE PROJECT

https://github.com/cdiener/pytest_tutorial

Clone/download please ☺.

WHAT IS TESTING?

Writing code to verify that other code works.

Developer usually participates only in unit testing or integration testing.

UNIT TESTING

A unit test is a small fragment of code that tests one *functional unit* of our software.

What in the world is the *unit*? Depends on you, most people test *functions* or *classes*.

WHAT IS A GOOD UNIT TEST?

A good unit test…

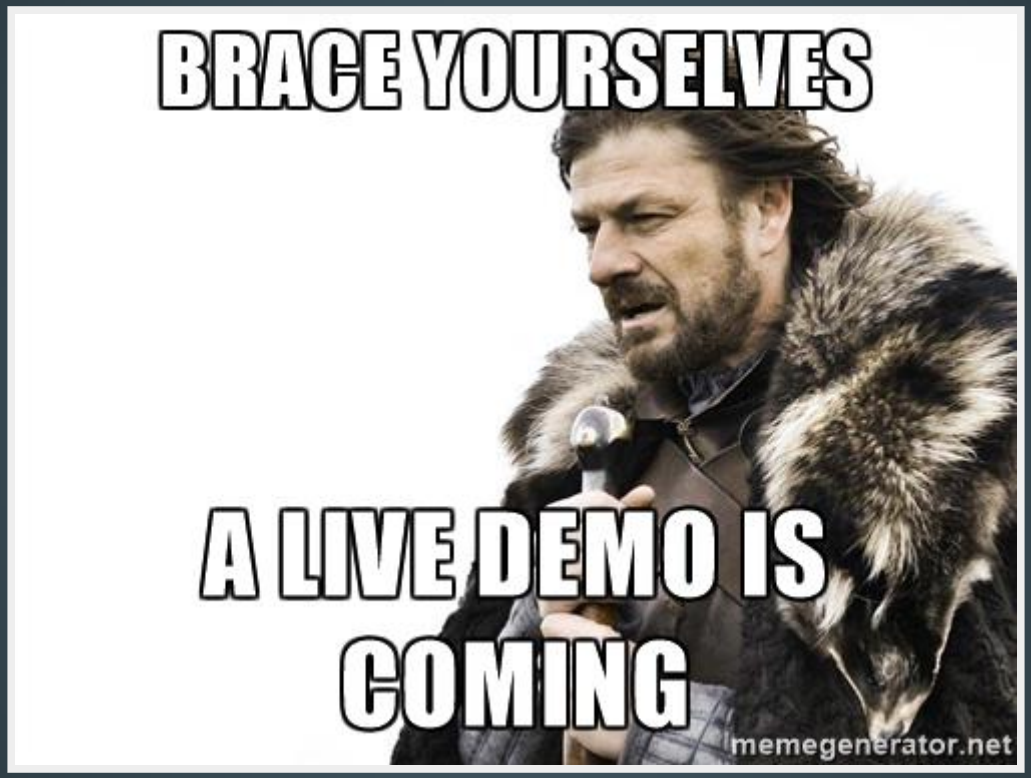
- isolates as much as possible
- can be scripted on paper
- tests edge cases
- is fast
- is not alone

There are limits to what a unit test can do though…

“No amount of testing can prove a software right, a single test can prove a software wrong.”
- Amir Ghahrai

Ok got it!

BUT HOW DO I WRITE UNIT TESTS?


BRACE YOURSELVES
A LIVE DEMO IS COMING
memegenerator.net

```
git clone https://github.com/cdiener/pytest_tutorial
cd pytest_tutorial

pip install --user flask pytest pytest-cov   # or pip3 ...
pip install --user  -e .
```

RUNNING PYTEST

Just running `py.test` in your package directory is ususall sufficient.

pytest will discover tests with the following rules:

- filename must start with `test_*`
- if the test is a function it also must start with `def test_*`
- if the test is a class it should start with `class Test*`

WRITING UNIT TESTS FOR PYTEST

A pytest unit test is simply a function that includes an `assert` statement.

```
def test_something():
    x = 2 + 3
    assert x > 2 and x > 3
    assert x == 5
```

FIXTURES AND SET UP

A fixture is a function that creates a configured object and can be injected into test functions.

```
import pytest
...

@pytest.fixture
def test_app():
    app = AppClass( "option 1", value=3)
    app.secret_key =  "test_key"
    app.validate()
    app.build_cache( 2000)
    return app


def test_app(test_app):
    response = test_app.get( "/rest/api/1234" )
    assert response
```

Some cool builtin fixtures:

- `capsys`: capture output to stdout/stderr
- `monkeypatch`: modify/delete system functions temporarily
- `tmpdir`: creates temporary directories and files
- `benchmark`: high accuracy benchmarking (requires `pytest-benchmark`)

ITERATING OVER PARAMETERS

Sometimes we want to run a test with several different inputs.

```
import pytest

@pytest.mark.parametrize("string", ["name123", "123 456", "ab123cd"])
def test_find_123(string):
    assert "123" in string
```

CHECKING FOR EXCEPTIONS

Sometimes we expect code to raise an Exception.

```
import pytest

def test_raises_error():
    with pytest.raises(ZeroDivisionError):
        a = 1/0
```

TEST COVERAGE

Coverage is the percentage of your code that is covered by tests.

There is no need to get 100% of coverage but it might help to identify sections of your code that are not well tested.

Many big companies go for coverage in the ninety-ish area.

If you have `pytest-cov` installed you can get coverage output using the `--cov` flag.

```
py.test --cov=networker

# with HTML output
py.test --cov=networker --cov-report=html

# To see missed lines afterwards
# might be coverage3 on your system
coverage report -m
```

To exclude files from coverage reports add the following section to your `setup.cfg` file.

```
[coverage:run]
omit = tests/*, */__main__.py
```

TEST COVERAGE AND CI

You can use https://travis-ci.org and http://codecov.io to generate coverage reports from your builds.

```
language: python
cache: pip
sudo: false

python:
    - 2.7
    - 3.5

before_install:
    - pip install -U pip
    - pip install -U pytest pytest-cov codecov

install:
    - pip install -e .

script:
    - py.test --cov=networker

after_success:
    - codecov
```

OK, WE KNOW PYTEST NOW!

Questions/comments?

 cdiener.com cdiener @thaasophobia