Given this, assert that:

fluent testing using fixtures and properties

PyCascades 2019

Paul Watts

 @paulcwatts  @joulespersecond

Who am I?

Reducing boilerplate using pytest fixtures


pytest

pytest is a test runner and framework
for creating better tests

pytest **fixtures** are functions
to help you reuse test setup logic

Example of a fixture

```
import pytest

@pytest.fixture
def smtp_connection():
    import smtplib
    return smtplib.SMTP("smtp.gmail.com", 587, timeout=5)

def test_ehlo(smtp_connection):
    response, msg = smtp_connection.ehlo()
    assert response == 250
```

Real world example

Testing API methods

```python
class WidgetTestCase(APITestCase):
    def test_get(self):
        """
        Test you can get a widget.
        """
        widget = Widget.objects.create(name="my widget")
        user = User.objects.create_user("test")
        self.client.force_authenticate(user)

        response = self.client.get(f"/api/widgets/{widget.pk}")

        self.assertEqual(response.status_code, 200)
        detail = response.json()
        self.assertEqual(detail["name"], "my widget")
```

AAA Pattern: Arrange, act, assert

https://simplabs.com/blog/2017/09/17/magic-test-data.html

```python
class WidgetTestCase(APITestCase):
    def test_get(self):
        """
        Test you can get a widget.
        """
        widget = Widget.objects.create(name="my widge
        user = User.objects.create_user("test")
        self.client.force_authenticate(user)

        response = self.client.get(f"/api/widgets/{widget.pk}"

        self.assertEqual(response.status_code, 200)
        detail = response.json()
        self.assertEqual(detail["name"], "my widget")
```

👈 Arrange

👈 Act

👈 Assert

Converting to pytest

Create our fixtures (arrange)

```
@pytest.fixture
def api_client() -> APIClient:
    return APIClient()

@pytest.fixture
def django_user(django_user_model) -> User:
    return django_user_model.objects.create_user("test")

@pytest.fixture
def auth_client(
    api_client: APIClient, django_user: User
) -> APIClient:
    api_client.force_authenticate(django_user)
    return api_client

@pytest.fixture
```

Our converted test

```
def test_get(auth_client: APIClient, widget: Widget) -> None:
    response = self.client.get(f"/api/widgets/{widget.pk}")

    assert response.status_code == 200
    detail = response.json()
    assert detail["name"] == widget.name
```

```python
def test_get(auth_client: APIClient, widget: Widget) -> None:
    response = self.client.get(f"/api/widgets/{widget.pk}")

    assert response.status_code == 200
    detail = response.json()
    assert detail["name"] == widget.name
```

☝️ Arrange

👉 Act

👉 Assert

Lessons from converting an entire codebase

- Not all setup can be fixtures

- Not all setup can be fixtures
- You may not get fewer total lines of code

- Not all setup can be fixtures
- You may not get fewer total lines of code
- Your tests get more focused

- Not all setup can be fixtures
- You may not get fewer total lines of code
- Your tests get more focused
- Mocks and fixtures: 👌

- Not all setup can be fixtures
- You may not get fewer total lines of code
- Your tests get more focused
- Mocks and fixtures: 👌
- `pytest.mark.parametrize` 💪

Property-based testing

```
def test_get(auth_client: APIClient, widget: Widget) -> None:
    response = self.client.get(f"/api/widgets/{widget.pk}")

    assert response.status_code == 200
    detail = response.json()
    assert detail["name"] == widget.name
```

```
def test_get(auth_client: APIClient, widget: Widget) -> None:
    response = self.client.get(f"/api/widgets/{widget.pk}")

    assert response.status_code == 200
    detail = response.json()
    assert detail["name"] == ""
```

```
def test_get(auth_client: APIClient, widget: Widget) -> None:
    response = self.client.get(f"/api/widgets/{widget.pk}")

    assert response.status_code == 200
    detail = response.json()
    assert detail["name"] == "絚\U000c51e1.\U0007c6f3狆/"
```

Hypothesis

https://hypothesis.works/

You decide what guarantees your code should make

You decide what guarantees your code should make

Express those as tests

You decide what guarantees your code should make

Express those as tests

Let hypothesis generate possible cases

A real world example

```python
def batcher(seq: Sequence[T], n: int) -> Iterator[Sequence[T]]:
    """
    Collect data into fixed-length chunks or blocks
    """
    start = 0
    slice = seq[start:n]
    while slice:
        yield slice
        start += n
        slice = seq[start : start + n]
```

Given the function's input,
what can we say about its output?

What guarantees do we want to check?

```
def test_batcher() -> None:
    """
    Given a list, it returns the same number of
    items in that list.
    """
```

```
from hypothesis import given, strategies as st

@given(seq=st.lists(st.integers()))
def test_batcher(seq: List) -> None:
    """
    Given a list, it returns the same number of
    items in that list.
    """
    result = batcher(seq, 2)
    total = sum(len(batch) for batch in result)
    assert len(seq) == total
```

```
widget/tests/test_utils.py::test_batcher Trying example: test_bat
Trying example: test_batcher(seq=[26330,
 -2113,
 -7680742127399325684,
 16759,
...
 14791])
Trying example: test_batcher(seq=[])
Trying example: test_batcher(seq=[21629, 118])
Trying example: test_batcher(seq=[3084, -23448, -2122764798, -85,
Trying example: test_batcher(seq=[-16460, 1911939345])
Trying example: test_batcher(seq=[-11521, 22559, -1209160822, 213
Trying example: test_batcher(seq=[])
Trying example: test_batcher(seq=[-32228])
Trying example: test_batcher(seq=[13282, -19446, -34, -40, 7345,
Trying example: test_batcher(seq=[-31486, -13218])
```

```
def test_batcher_inverse() -> None:
    """
    Given a list, it returns the items in that list.
    """
```

```
from hypothesis import given, strategies as st
from itertools import chain

@given(seq=st.lists(st.integers()))
def test_batcher_inverse(seq: List) -> None:
    """
    Given a list, it returns the items in that list.
    """
    result = batcher(seq, 2)
    flat = list(chain.from_iterable(result))
    assert seq == flat
```

Given this, assert that

"propositional calculus"

```
def test_get(auth_client: APIClient, widget: Widget) -> None:
    response = self.client.get(f"/api/widgets/{widget.pk}")

    assert response.status_code == 200
    detail = response.json()
    assert detail["name"] == widget.name
```

```
def test_get() -> None:
    """
    Given a widget and an authorized user,
    you can GET it from the API
    """
```

```
def test_get(auth_client: APIClient, widget: Widget) -> None:
    """
    Given a widget and an authorized user,
    you can GET it from the API
    """
```

```
def test_get(auth_client: APIClient, widget: Widget) -> None:
    """
    Given a widget and an authorized user,
    you can GET it from the API
    """
    response = self.client.get(f"/api/widgets/{widget.pk}")

    assert response.status_code == 200
    detail = response.json()
    assert detail["name"] == widget.name
```

```python
from hypothesis import given
from .strategies import widgets

@given(widget=widgets())
def test_get(auth_client: APIClient, widget: Widget) -> None:
    """
    Given a widget and an authorized user,
    you can GET it from the API
    """
    response = self.client.get(f"/api/widgets/{widget.pk}")

    assert response.status_code == 200
    detail = response.json()
    assert detail["name"] == widget.name
```

Questions

What if I don't use pytest?

Do I have to convert all my tests
to property-based tests?

Next steps

Hypothesis for:

data science

mocking

Hypothesis and contracts:

https://hillelwayne.com/talks/beyond-unit-tests/

How to come up with properties:

https://fsharpforfunandprofit.com/posts/property-based-testing-2/

Caveat:

https://github.com/pytest-dev/pytest/issues/916

Acknowledgements

Mariatta Wijaya

Don Sheu

Dustin Ingram

Russell Duhon

Cris Ewing

Alan Vezina

Maelle Vance

Thank you