PRACTICAL TESTING

WITH PYTEST

ASPP APAC 2019

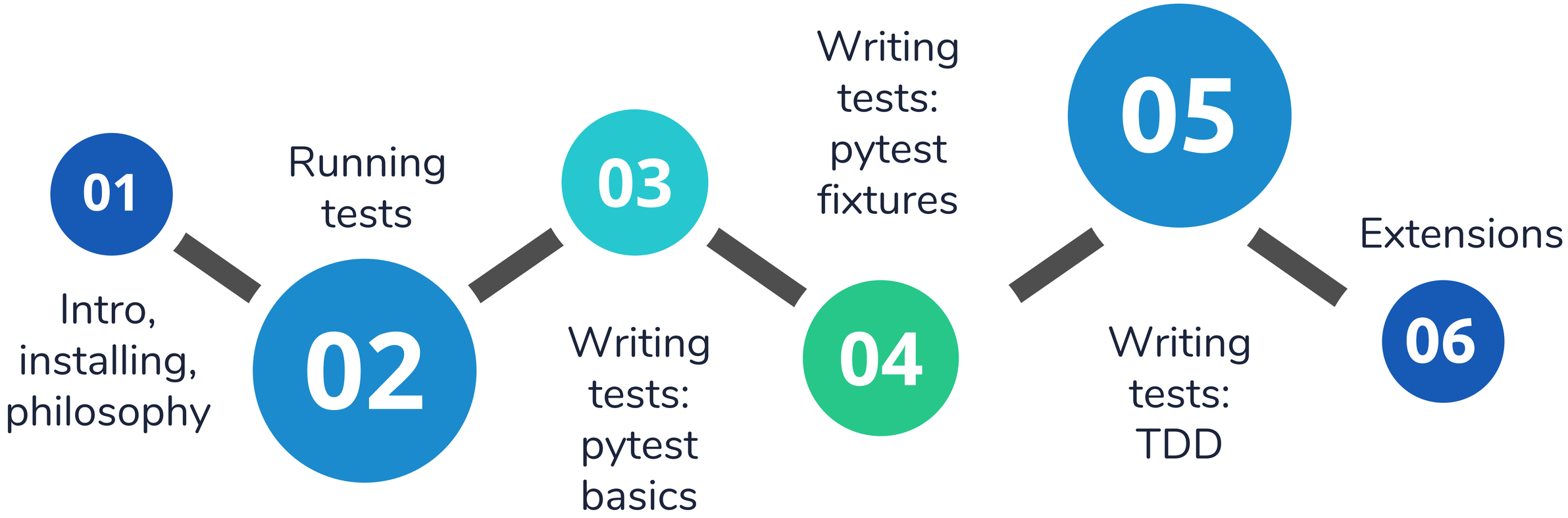
BRIANNA LAUGHER

Links!

Repo with all code and notes:

`https://github.com/aspp-apac/2019-testing-code`

OUTLINE


01
Intro, installing, philosophy
02
Running tests
03
Writing tests: pytest basics
04
Writing tests: pytest fixtures
05
Writing tests: TDD
06
Extensions

INTRO, INSTALLING, PHILOSOPHY

1

Hello!

I am Brianna Laugher

Pytest user for 6+ years
Contributor and evangelist

twitter/github/slack @pfctdayelise

WHY ARE WE HERE?

WHAT IS A TEST?

What is a test?

```
import datetime


def test_what_month_is_it():
    today = datetime.datetime.today()
    assert today.month == 1, "Isn't it January?"
```

What is a test?

```
import datetime


def test_what_month_is_it():
    today = datetime.datetime.today()
    assert today.month == 7, "Isn't it July?"
```

assert keyword

space

condition (evaluated to Boolean)

comma

message (a string)

these two are optional (you can leave them off)

Assert statement

```
assert today.month == 7, "Isn't it July?"
```

Is basically the same as

```
if not(datetime.datetime.today().month == 7):
    raise AssertionError("Isn't it July?")
```


WHY
TESTING

TESTING PHILOSOPHY

Sarah Mei's "Five Factor Testing":

- Verify the code is working correctly
- Prevent future regressions [bugs that come back]
- Document the code's behavior
- Provide design guidance
- Support refactoring

Joep Schuurkes
@j19sch

Follow

The question "Why is testing taking so long?" is not dissimilar to a restaurant manager asking:
Why are we taking so much time to check if the food is not over- or undercooked, over- or underseasoned, that the flavors are in balance? The food has been prepared, serve it already!

12:23 AM - 7 Jan 2019


WHY
PYTEST

Hynek Schlawack
@hynek

attrs SLOC: src: 895, tests: 1540

If you use bad testing tools, you write the majority of your software with bad tools. #pytest #hypothesis

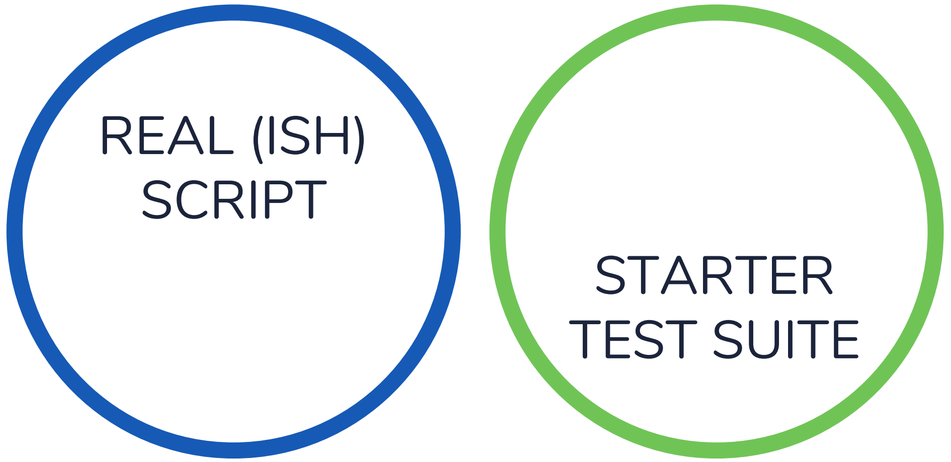
10:35 PM · Mar 4, 2017 from City Centre, Berlin · Tweetbot for Mac

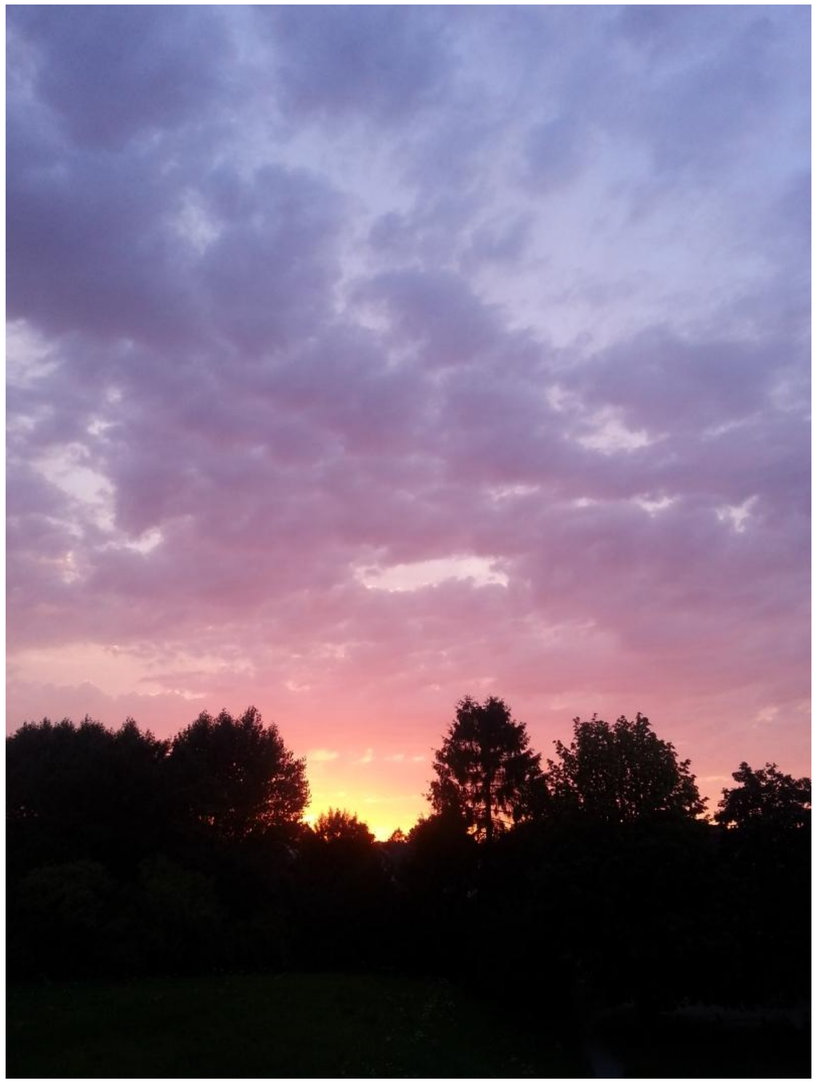
Stephanie Hyland
@_hylandSL

tried writing unit tests (@pytestdotorg)
got weirdly into it
experiencing a novel sense of actually trusting my code

10:08 PM · Jun 1, 2017 · Twitter Web Client

TODAY





Dominant Color

colorThief.getColor(image):158ms

Palette

colorThief.getPalette(image):424ms

ACTIVITY

- [ ] git clone `https://github.com/aspp-apac/2019-testing-code`

  (or sync your fork with `git pull` if you cloned it previously)

- [ ] Make a virtualenv with Python 3.6

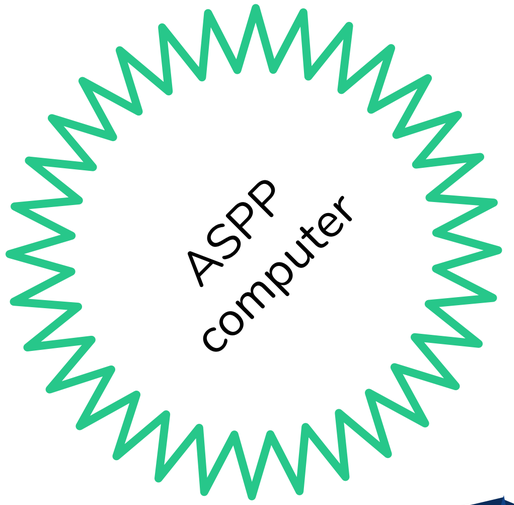
- [ ] Install requirements and run script locally

- [ ] Practice writing assert statements

- [ ] Explore as a user in your terminal (exploratory testing!)

ACTIVITY

- `git clone https://github.com/aspp-apac/2019-testing-code`
  `cd 2019-testing-code`
- `source activate aspp`
  `pip install -e .`
- Run script locally: `python colorthief.py images/sunset.jpg --fancy`
- Practice writing assert statements
- Explore as a user in your terminal (exploratory testing!)



RUNNING TESTS

2

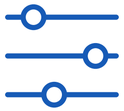
Exploring the pytest test runner

pytest --help

My favourites:

--collectonly
Don’t execute any tests, just print their names

-k FOO
Only collect/execute tests that match FOO in their path/name.



-x, --exitfirst
Stop executing tests after the first failure.

-s
Show stdout (print statements) even in passing tests

--tb=short
Show a short traceback.Other options: auto/long/short/line/native/no

-v, --verbose
More verbose output (eg with test function names not just ‘.’)

pytest --help #2

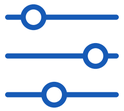
Even more:

--lf, --last-failed

Rerun only the tests that failed at the last run (or all if none failed)

-l, --showlocals

Show locals in traceback when a test fails. (Who needs print statements!)



--junit-xml=path/to/file.txt

Create JUnit-style XML report file (useful for eg Jenkins).

-rxs

Show reasons for xfailed and skipped tests.

--strict

Python warnings become errors (ie cause a test to fail).

--pdb

Launch pdb (Python debugger) when a test fails. (Best used with -x)

ACTIVITY

- Run `pytest`
  or (if you see import errors)
  `python -m pytest`

  What output do you see? Does everything pass?

- Run `pytest --help`
  or
  `python -m pytest --help`

- Try to find what options you need to do the following:
  - only run the ‘fancy_output’ tests
  - get a code coverage report
  - get a report on the slowest tests

WRITING TESTS: BASICS

3

Decorators

The @ symbol applies a decorator to the function/method that follows. The decorator “wraps” the function and has many different uses.

```
@pytest.mark.xfail()
def test_what_month_is_it():
    today = datetime.datetime.today()
    assert today.month == 1, "Isn't it January?"
```

How does pytest know what is a test?

By default:

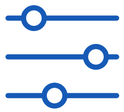
- Filename: `test_foo.py` or `foo_test.py`
- Function: name `test_foo`
- Class: `TestFoo`, with no `__init__` method
  - Methods: name `test_foo`

Classes are usually for grouping related tests, pytest does not require them. Feel free to just write test functions.

BASIC FEATURES

@pytest.mark
Ad-hoc tags to label or group tests

@pytest.mark.skipif
Conditions to skip tests



@pytest.mark.parametrize
Feeding different input/output through the same test steps

@pytest.mark.xfail
When a test is expected (required) to fail

@pytest.raises
Expecting exceptions

pytest.ini, conftest.py
Configuration, fixtures

Pytest test result types

- . PASS: No exception raised during test execution
- F FAIL: An exception was raised during execution

- s SKIP: Test not executed as it was marked to be skipped
  - e.g. *test is for Windows, but currently environment is Linux*

- x XFAIL: Test was marked as “expected to fail”, and it did fail
- X XPASS: Test was marked as “expected to fail”, but it passed!

ACTIVITY

- Look at colorthief.py and test_palettes.py
- Fix a failing test!
- Add a new test!
- Explore marks, parametrize, raises...

WRITING TESTS: FIXTURES

4

Typical parts of a test

Given (some preconditions)

When (some action)

Then (some result is expected)

*[and some **cleanup** might be needed]*

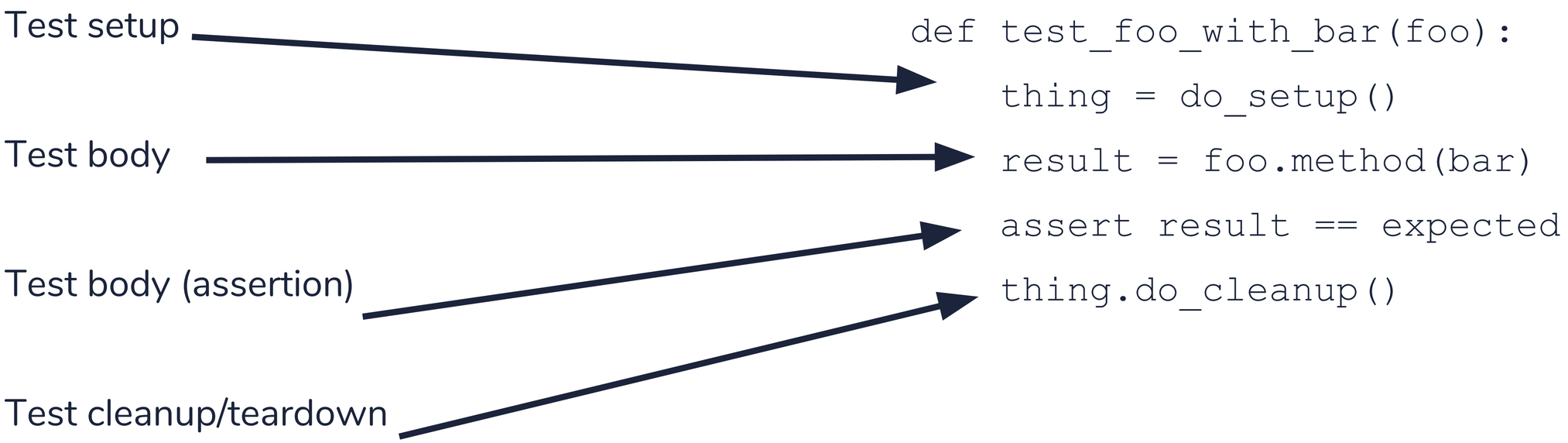
Test setup

Test body

Test body (assertion)

Test cleanup/teardown

Naive test structure



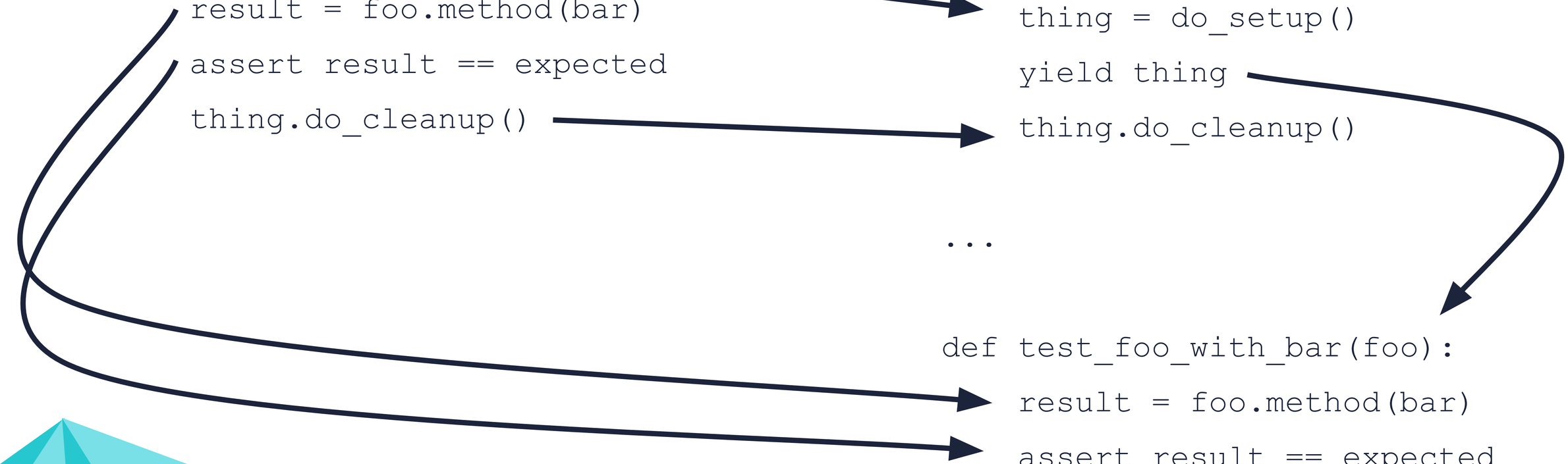
Test structure with pytest fixtures

```
def test_foo_with_bar(foo):
    thing = do_setup()
    result = foo.method(bar)
    assert result == expected
    thing.do_cleanup()
```

```
@pytest.fixture
def foo():
    thing = do_setup()
    yield thing
    thing.do_cleanup()

...

def test_foo_with_bar(foo):
    result = foo.method(bar)
    assert result == expected
```



What is a fixture?

It’s some resource your test needs to run.

Could be some data in Python, or in a database, or the database itself.

In Django it tends to mean “json of data to feed into database for this test”, but in pytest it can be broader.

Often have a notion of ‘setup’ and ‘teardown’/’cleanup’.

In pytest, they have a defined scope - function, class, module, session -and they can be built on top of each other.

Fixtures separate the set-up of your test from your actual test code, making it easier to “get to the point”.

A fixture in action

```
def test_analyse_image_not_fancy(capsys):
  imgpath = 'images/sunset.jpg'
  analyse_image(imgpath, do_print_fancy=False)
  expected_stdout = """(163, 143, 178)
(9, 6, 5)
(99, 35, 32)
(246, 222, 171)
(151, 82, 64)
"""
  captured = capsys.readouterr()
  assert captured.out == expected_stdout
```

PYTEST/PLUGIN FIXTURES

monkeypatch

Provides helper methods to modify objects, dictionaries or os.environ. All modifications will be undone after the test function or fixture has finished.

caplog

Supplied by pytest-catchlog plugin. Useful for testing what gets written to logs.

tmpdir

Return a temporary directory path object which is unique to each test function invocation.

freezer

Supplied by pytest-freezegun plugin. Useful for “freezing” datetime.

browser

Supplied by pytest-splinter plugin. Useful for controlling a browser to test web applications.

responses

Supplied by pytest-responses plugin. Mock calls to requests when methods call external APIs.

Avoiding external services...where?

Outside Python

Create a lightweight ‘thing’, use test config to point to that

e.g. SQLite database

Test mail/web server

(**pytest-localserver**)

Inside Python

Use monkeypatch to replace specific methods

e.g. datetime

Can be very tricky to get right, may cause side effects in other libraries

Inside your app

Use DI or monkeypatch (maybe with mock) to replace the bits your application cares about

Your module.py does ‘import X’, so only monkeypatch the X in your module

ACTIVITY

- Examine conftest.py
- Write your own fixture!

WRITING TESTS: TDD

5

Test driven development

fiona
@fioroco

Follow

who called it test-driven development and not asserts before reverts

5:44 AM - 16 Sep 2015

38 Retweets **56** Likes

1 38 56

“I write to

find out

what I’m thinking”

“I write tests to

find out

how my code should work”

TDD: Red, Green, Refactor

Want to write a new method or function?

- Write a test.
  - Run the test.
    - It obviously fails (red).
- Write the simplest possible implementation that makes the test pass.
  - No optimising. Could be hardcoded return value.
    - Run the test, now it passes (green).
- Now, rewrite the code so you are happy with it.
  - Repeatedly re-run the test, ensuring it stays passing.
    - You have successfully refactored! And you have the confidence of knowing it still does what it needs to do.

Dependency injection

```
class Foo:
    def __init__(self):
        self.bar = Bar()
        self.baz = Baz()

    def do_thing(self):
        ...


...
f = Foo()
result = f.do_thing()
```

```
class Foo:
    def __init__(self, bar, baz):
        self.bar = bar
        self.baz = baz

    def do_thing(self):
        ...


...
f = Foo(Bar(), Baz())
result = f.do_thing()
```

Dependency injection - before

```
import datetime



def days_to_birthday(dob):
  birthday = datetime.date(today.year, dob.month, dob.day)
  today = datetime.date.today()
  if birthday < today:
      birthday = datetime.date(today.year + 1, dob.month, dob.day)
  return (birthday - today).days
```

Dependency injection - after

```
import datetime



def days_to_birthday(dob, today=None):
  birthday = datetime.date(today.year, dob.month, dob.day)
  today = today if today is not None else datetime.date.today()
  if birthday < today:
      birthday = datetime.date(today.year + 1, dob.month, dob.day)
  return (birthday - today).days
```

Dependency injection

patrick thomson
@importantshock

Follow

I love to use[1] dependency[2] injection[3]

[1] pass
[2] values
[3] to functions

7:28 PM - 16 Jan 2019

Options

MONKEYPATCH/MOCK

- When you want to get test coverage on legacy code you can't change, or before you refactor it
- Safely replace a method/attribute on the 'real' object
- "Safely" = monkeypatch fixture will clean up after itself

DEPENDENCY INJECTION

- When you do TDD, can refactor your code, or it's nicely designed to start with!
- Could be 'injecting' a 'real' thing, or a monkeypatched 'real' thing, or a mock object

Too much monkeypatch, or too much mock, can make your tests brittle

DI with lightweight objects can miss bugs where those objects differ to the 'real' ones. eg SSL

Other design considerations

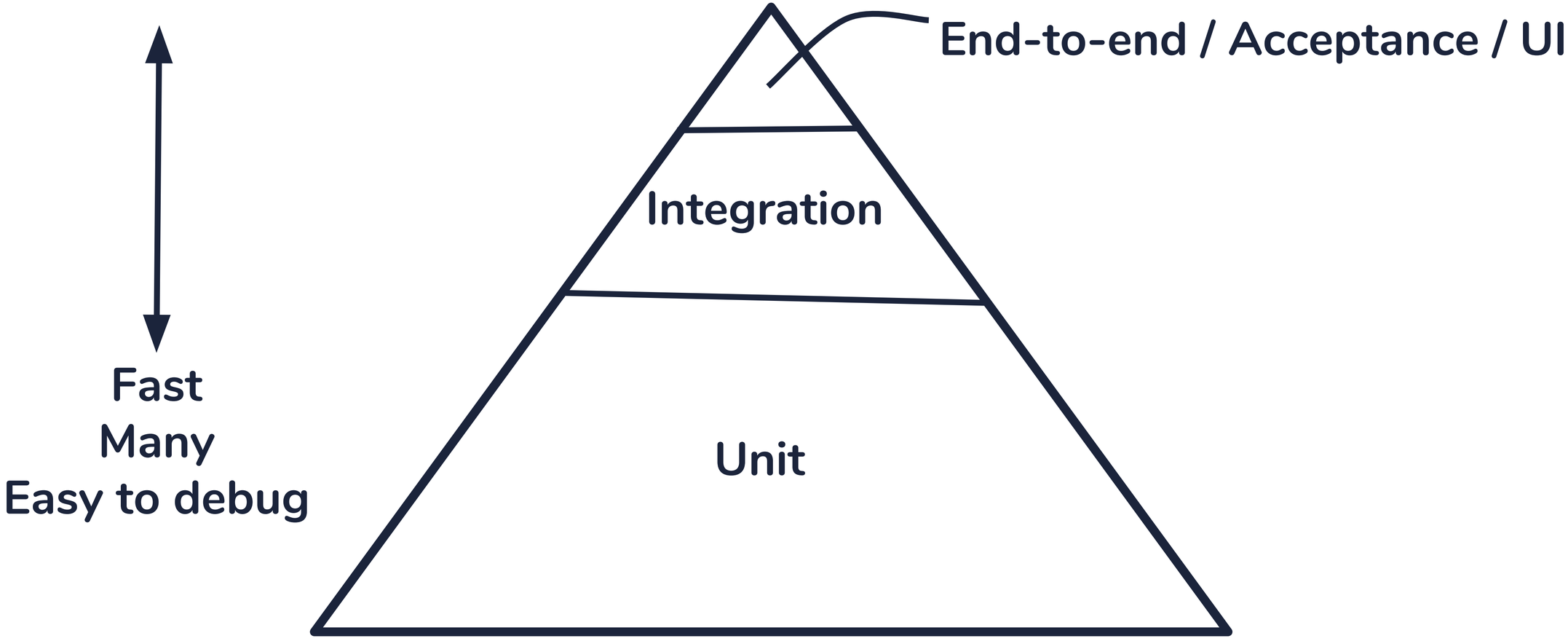
Functions that do one thing

- Or methods
- Or classes
- Or modules
- and so on :)

Side effects

- Printing
- Logging
- Reading files
- Writing files

The testing pyramid



ACTIVITY

- Let’s do some TDD!
- Add tests, and then command-line options, to specify quality and color_count
- Add tests, and then a command-line option, to output hex values instead of RGB

EXTENSIONS 6

CI, IDE, debugging, plugins

Plugins

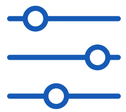
Plugins can modify test collection, execution or output. They often give you new command-line options or fixtures. A good starting point: `plugincompat.herokuapp.com`

pytest-xdist

Run tests in parallel / distributed.

pytest-sugar

Progress bar and enhanced output from running tests.



your fave framework

django,
flask,
twisted,
pyramid,
aiohttp,
asyncio,
qt...

pytest-pep8

Causes a test run to fail if a source file violates the PEP8 style guide.
Also pytest-pylint, pytest-flakes

pytest-catchlog

Make assertions on log contents

What is Continuous Integration?

“The tests must run, and pass, on every commit to a source code repository”

A CI server enforces this!

It keeps your test suite passing.

Jenkins, Travis CI, Circle CI, Azure Pipelines, Gitlab CI, Bamboo (Atlassian), ...

Travis CI can be easily configured/integrated with your Github repository.

ACTIVITY

- What do you think would be difficult to test? See if there is a pytest plugin for it.
- Set up Travis CI for your fork
- Look for a plugin to run pytest from your IDE
- Explore numpy.testing helper methods for assertions about arrays

Thanks!

Any questions?

You can find me at: @pfctdayelise
brianna@laugher.id.au

SLIDES CREDITS

Special thanks to all people who made and share these awesome resources for free:

- Presentation template designed by Slidesmash
- Vector Icons by Matthew Skiles

Presentation Design

This presentation uses the following typographies and colors:

Free Fonts used:

https://www.fontsquirrel.com/fonts/nunito

Colors used