

Preparation

- Load up the VM in VirtualBox
- Open a terminal window
- Move to directory `~/williams`
 - `source conda.sh`
 - `conda activate intel_env`
- Move to the directory `~/williams/python_testing`
 - `git pull`
 - `pip install -r requirements.txt`



pytest

Make testing easy

Matt Williams, University of Bristol

Test runner

- pytest comes with a flexible *test runner* which can find your tests
- Searches inside files called `test_*.py` or `*_test.py` for functions prefixed with `test_`
- Once collected, runs them all and prints results
- All is configurable, can also run doctests, different prefixes, extra plugins etc.
- Look at `pytest --help` for more information

Write a test function

- Name the file and function correctly
- A test *passes* if the function finishes running without error
- A test *fails* if any exception is uncaught
- pytest overrides Python's `assert` statement to make it more useful

First test

No need to import any pytest module

```
from my_lib import add_elements
```

Start name with 'test_'

```
def test_add():  
    assert add_elements([1, 2], [3, 4]) == [4, 6]
```

Nothing explicitly pytest-specific in the source code

Copy this out into `test_my_lib.py` and run `pytest`.
Try breaking the test to see what failures look like.
Try running with `pytest -v` for more info

Morse code

- There is a file `morse.py` in the directory.
- Make a new test file which contains two test functions:
 - One for each function in `morse.py`
 - Each test only needs to check a few letters
 - Make sure you use an `assert` in each
 - Run them with `pytest` on the command line
 - Make sure they *can* fail
 - Tip: `"sos"` → `"... --- ..."` and vice versa

Parametrising tests

- Often you want to run a test with lots of different inputs
- Want an individual pass/fail for each
- Can use a special pytest decorator with a list of arguments
- We'll start by translating previous example to a parametrised form

Parametrised test

```
import pytest
from my_lib import add_elements

@pytest.mark.parametrize('a,b,answer', [
    ([1, 2], [3, 4], [4, 6]),
])
def test_add(a, b, answer):
    assert add_elements(a, b) == answer
```

Explicitly using the module this time so must import it

Name the arguments in a string

Refer to those names in your test function

They will contain the specified values each time the test is run

Copy this out and run pytest.

Add a few more parameter tuples and run pytest again.

Try changing the argument names (answer, a and b)

Catching exceptions

- Sometimes you *want* your code to 'fail'
- Test for expected exceptions with a `pytest.raises` context manager
- Given the name of an exception, it will only pass if that exception is raised

Catching exceptions

```
...  
def test_wrong_type():  
    with pytest.raises(TypeError):  
        add_elements([1, 2], 6)
```

Give the name of the exception that you expect to be raised

Any exceptions raised in the block will be processed

Copy this out and run pytest.

What happens if the exception is *not* raised in the block?

Set up with fixtures

- Fixtures allow reusable set-up and tear-down of test environments
- Useful for complex or heavy setup logic
- They can be reused between tests
- Many can be combined and layered together

Simplest dummy fixture

```
import pytest
from my_lib import add_elements

@pytest.fixture
def pair_of_lists():
    return ([1, 2], [3, 4])

def test_add(pair_of_lists):
    list1 = pair_of_lists[0]
    list2 = pair_of_lists[1]
    assert add_elements(list1, list2) == [4, 6]
```

Use the decorator to turn this function into a fixture

Return value is used as value of fixture

Refer to fixture by name.
Return value is assigned to variable.

First useful fixture

```
import lzma
import pytest
from my_lib import get_gutenberg_text, word_count
```

Use the decorator to make this function a fixture

```
@pytest.fixture
def warandpeace():
    with lzma.open('warandpeace.txt.xz', mode='rt') as f:
        text = f.read()
    book_text = get_gutenberg_text(text)
    return book_text
```

Return value is used as value of fixture

Refer to fixture by name.

Return value is assigned to variable.

```
def test_count_lines(warandpeace):
    assert len(warandpeace.split('\n')) == 6567
```

Copy this out and run `pytest`.

What happens if you misspell the fixture name?

Mocking

- Mocking allows you you to fake-up the environment the test runs in
- We can do this in pytest using the `pytest-mock` package
- Useful if you are testing something with side-effects
- Also useful for slow code
- Should complement integration tests, not replace them
- Require knowledge of the internals of the thing you're testing

Mocking

`my_lib` has a function which downloads from the internet using `requests`. We don't want this happening in the test suite as it might be flaky and slow. We can use a mock to fake the internet part.

```
from my_lib import count_capital_words_in_website as cw
from types import SimpleNamespace as NS
```

```
def test_count_words(mocker):
    requests_get = mocker.patch("requests.get",
                                return_value=NS(text="Set of words With Capitals"))
    assert cw("http://example.com") == 3
    requests_get.assert_called_once()
```

Use the mocker fixture from pytest -mock

patch takes name of function to mock

We can specify the fake return value

We can assert that the function was called

Copy this out and run `pytest`.

Change the `return_value` to make sure it works as you expect

Mocking

- Look at `read_databases.py`
- We have a function (`read_database`) which calls other functions (`BAH`, `NMHD`, `LPMS`)
- We want to make sure it does what it says
- Starting from what is in `test_read_databases.py`, mock away the calls to `BAH`, `NMHD` and `LPMS` and check that they *would* have been called.
- Can we check what arguments they would be called with?
 - Tip: behind the scenes, `pytest-mock` uses Python's `unittest.mock`

Auto-generate tests

- With our first parametrized test we gave an explicit list of examples
- *Hypothesis* can generate examples for us
- We define the constraints and it generates a minimal failing example
- *Strategies* tell Hypothesis how to generate examples

Hypothesis

Strategies define how to generate examples



```
from hypothesis import given
from hypothesis.strategies import lists, integers

from my_lib import add_elements
```

@given() assigns strategies to arguments



```
@given(lists(integers()), lists(integers()))
def test_add(a, b):
    add_elements(a, b)
```



Each argument will be set to a list of integers

Copy this out and run pytest.
Can you fix the bug in my_lib.py?

Hypothesis - Morse

```
from hypothesis import given
from hypothesis.strategies import lists, sampled_from

from morse import encode, decode, letter_to_morse

@given(We'll pass in lists..
       lists(... of elements samples from...
             elements=sampled_from(
                 list(letter_to_morse.values()))
             ... our Morse letters.
             )))
def test_roundtrip_morse(morse):
    morse = ' '.join(morse)
    assert encode(decode(morse_message)) == morse
    Check the round-trip
```

Copy this out and run pytest.

Add a second test which checks English → Morse → English
(tip: check the `string` Python module and <https://v.gd/tgyKOL>)

Do you find a bug/missing feature in `encode`?

Hypothesis - pandas

- Read the code in `analyse_weather.py`
- We want to test the `hottest_summer` function.
- Using Hypothesis, write a test based on the code below which fully tests the advertised interface
- You'll need the Hypothesis docs for [Pandas](#) and [general strategies](#)

```
from hypothesis import given
from hypothesis.extra.pandas import columns, \
    data_frames, range_indexes
import hypothesis.strategies as st
import pandas as pd
from analyse_weather import hottest_summer

@given(...)
def test_hottest_summer_auto(df):
    assert not pd.isnull(hottest_summer(df))
```

What's next

- Other pytest functions like `pytest.approx` and `pytest.mark.skipif`
- Doctests (see examples in `morse.py`)
- Useful built-in fixtures like `tmpdir`
- Plugins like `pytest-cov` for coverage analysis
- Start writing tests!