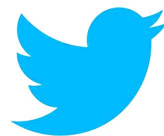
pytest

Automated testing in python

Ashwini Balnaves

- 1 year as a Quality Engineer
- Software Engineer working on network automation
- Software Engineer at Kapiche
- Twitter: @that_ashwini



Why write automated tests?

Why I love automated testing…

There are many reasons, but here’s two:

1. I get to develop against something that I know is stable.
2. I can lock in a contract of behaviour for the code.

So what is pytest?

There are others...

1. doctest
2. unittest
3. Nose
4. Domain Specific Language Frameworks (Robot)

Installing pytest

```
pip install -U pytest
```

How to write a test in pytest

pytest needs:

1. A file that is named `*_test.py` or `test_*.py`
2. A test to run
   - A function called `test_*` or
   - A class named `Test*` which has a method called `test_*`
3. A condition to test

```python
# my_project/my_project_test

def test_case_one():
    assert True
```

```python
# my_project/my_project_class_test.py

class TestThreeCases():

    def test_case_one(self):
        assert True

    def test_case_two(self):
        assert True

    def test_case_three(self):
        assert False
```

How to run pytest

1. From the command line

```
$ pytest
```

2. From a plug in in your IDE

```
(venv) Spares-MacBook-Air:my_project spare$ pytest -v
============================== test session starts ===============================
platform darwin -- Python 3.7.2, pytest-4.4.0, py-1.8.0, pluggy-0.9.0 -- /Users/sp
hon3.7
cachedir: .pytest_cache
rootdir: /Users/spare/my_project
collected 4 items

my_project_class_test.py::TestThreeCases::test_case_one PASSED           [ 25%]
my_project_class_test.py::TestThreeCases::test_case_two PASSED           [ 50%]
my_project_class_test.py::TestThreeCases::test_case_three FAILED         [ 75%]
my_project_test.py::test_case_one PASSED                                 [100%]

==================================== FAILURES ====================================
________________________ TestThreeCases.test_case_three _________________________

self = <my_project_class_test.TestThreeCases object at 0x109c13b38>

    def test_case_three(self):
>       assert False
E       assert False

my_project_class_test.py:7: AssertionError
====================== 1 failed, 3 passed in 0.13 seconds =======================
(venv) Spares-MacBook-Air:my_project spare$
```

Now we have done the basics

So many features!

- Functions
  - pytest.approx
  - pytest.fail
  - pytest.param
  - pytest.raises
  - pytest.warns
- Marks
  - pytest.mark.parametrize
  - pytest.mark.skip
  - pytest.mark.skipif
  - Pytest.mark.xfail
- Hooks
  - Bootstrapping hooks
  - Initialization hooks
  - Test running hooks
  - Collection hooks
  - Reporting hooks
  - Debugging/Interaction hooks
- Fixtures
  - @pytest.fixture
  - config.cache
  - doctest_namespace
  - request
  - pytestconfig
  - record_property
  - caplog
  - monkeypatch
  - testdir
  - tmp_path
  - tmpdir
- Environment Variables
  - PYTEST_ADDOPTS
  - PYTEST_DEBUG
  - PYTEST_PLUGINS
  - PYTEST_DISABLE_PLUGIN_AUTOLOAD
  - PYTEST_CURRENT_TEST

@pytest.mark.parametrize

Time to parametrize!

```
from my_project import my_sorter

def test_string_cba():
    input_data = ['c', 'b', 'a']
    expected = ['a', 'b', 'c']
    output = my_sorter(input_data)
    assert output == expected

def test_string_4231():
    input_data = ['4', '2', '3', '1']
    expected = ['1', '2', '3', '4']
    output = my_sorter(input_data)
    assert output == expected

def test_int_4231():
    input_data = [4, 2, 3, 1]
    expected = [1, 2, 3, 4]
    output = my_sorter(input_data)
    assert output == expected
```

```
from my_project import my_sorter
import pytest

@pytest.mark.parametrize('input_data, expected',[
    (['c', 'b', 'a'], ['a', 'b', 'c']),
    (['4', '2', '1', '3'], ['1', '2', '3', '4']),
    ([4, 2, 1, 3], [1, 2, 3, 4]),
])
def test(input_data, expected):
    output = my_sorter(input_data)
    assert output == expected
```

```python
from my_project import my_sorter
import pytest

@pytest.mark.parametrize('input_data, expected',[
    (['c', 'b', 'a'], ['a', 'b', 'c']),
    (['4', '2', '1', '3'], ['1', '2', '3', '4']),
    ([4, 2, 1, 3], [1, 2, 3, 4]),
    ([], []),
    ('orange', ['a', 'e', 'g', 'n', 'o', 'r']),
    ({5, 4, 2}, [2, 4, 5]),
])
def test(input_data, expected):
    output = my_sorter(input_data)
    assert output == expected
```

@pytest.fixture

pytest fixtures: explicit, modular, scalable

New in version 2.0/2.3/2.4.

The purpose of test fixtures is to provide a fixed baseline upon which tests can reliably and repeat-edly execute.

Reusable setup and teardown code

Available for functions, modules, sessions setup and and teardown

```python
from my_project import my_sorter
import pytest

@pytest.fixture
def my_resource():
    # code that runs BEFORE the test function
    print('before yield')
    # value that gets passed to the test
    yield 'return_value'
    # code that runs AFTER the test function
    print('after yield')


def test_my_sorter(my_resource):
    print('running test')
    print(my_resource)
    assert False
```

```
====================== test session starts ======================
platform darwin -- Python 3.7.2, pytest-4.4.0, py-1.8.0, pluggy-0
.9.0 -- /Users/spare/my_project/venv/bin/python3.7
cachedir: .pytest_cache
rootdir: /Users/spare/my_project
collected 1 item

my_project_sorter_test_fixture.py::test_my_sorter FAILED  [100%]

=========================== FAILURES ===========================
________________________ test_my_sorter ________________________

my_resource = 'return_value'

    def test_my_sorter(my_resource):
        print('running test')
        print(my_resource)
>       assert False
E       assert False

my_project_sorter_test_fixture.py:16: AssertionError
--------------------- Captured stdout setup ---------------------
before yield
--------------------- Captured stdout call ----------------------
running test
return_value
------------------- Captured stdout teardown --------------------
after yield
=================== 1 failed in 0.12 seconds ====================
```

```python
import pytest

@pytest.fixture(scope='session')
def session_resource():
    print('session before')
    yield
    print('session after')

@pytest.fixture(scope='module')
def module_resource():
    print('module before')
    yield
    print('module after')

@pytest.fixture(scope='function')
def function_resource():
    print('function before')
    yield
    print('function after')

def test_fixture_true(session_resource,
                      module_resource,
                      function_resource):
    assert True
    print('test one')

def test_fixtures_false(session_resource,
                        module_resource,
                        function_resource):
    assert True
    print('test two')
```

```
========== test session starts ==========
platform linux -- Python 3.5.6, pytest-4
.4.0, py-1.8.0, pluggy-0.9.0
rootdir: /home/ashwini/my_project
collected 2 items

fixture_test.py session before
module before
function before
test one
.function after
function before
test two
.function after
module after
session after


======= 2 passed in 0.01 seconds =======
```

```python
import pytest

@pytest.fixture(params=['file1.txt', 'file2.txt'])
def my_resource(request):
    with open(request.param) as f:
        yeild f
```

I’m in love with testing, now what?

- Explore the pytest plugins
- Integrate pytest into your development pipeline
- Check your test coverage (with pytest-cov)
- Fall in love with mocks - dependency injection for python