PyTest

Right Way to Test Python Code

Haroon Rashid
SEECS, NUST

@haroonrashid235

About Me

NUST
NATIONAL UNIVERSITY
OF SCIENCES & TECHNOLOGY

Google
Summer of Code

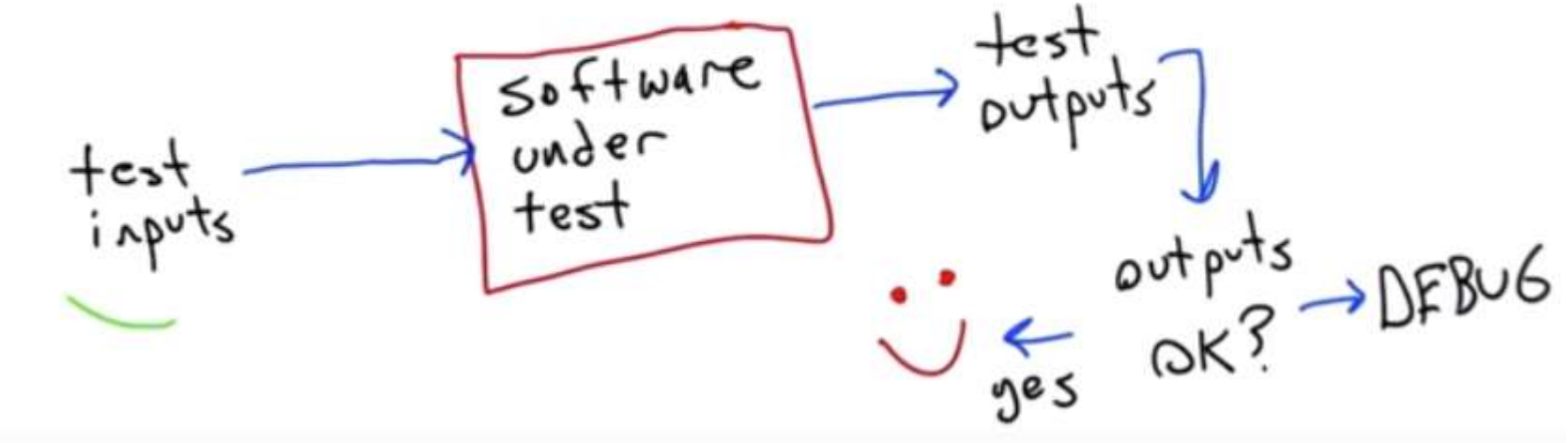
python
SOFTWARE FOUNDATION

Contents

- What is Software Testing?
- Why Test Code?
- Types of Testing
- Test Coverage
- Testing in Python
- Pytest and its Features
- Fixtures and parameterization

What is Software Testing?

- Evaluation of a software to make sure the specifications are met



(Software Testing - Udacity)

Why Test Code?

Testing Code increases:

- Trust
- Confidence

Why Test Code? - Continued

But…

- You can never be 100% confident
- In most cases, impossible to test entire input domain
- More you test, more confidence in the correctness of software

How many Tests to Write?

- No definitive answer
- 1000s of tests may fail to find a bug
- Smart Testing >> More and More Tests

Better Question.. How should I test it?

- Work as a Detective
- You are a bug hunter as a software tester
- Learn the Product/Software as much as possible
- Seek complexity underlying the simplicity of the code
- Try to learn **How might it not work!**
- Goal is to make software not work

Types of Testing

1. **Unit Testing** - Isolated modules e.g. method or a function
2. **Integration Testing -** Multiple software modules that are already unit tested
3. **System Testing -** Test with everything plugged together
4. **Acceptance tests** - Test the customer's use case

Some other types:

- Black Box Testing
- White Box Testing

Test Coverage

- Measures Proportion of program Executed during testing
- Score Metric as a percentage
- 100% Coverage doesn’t mean that all bugs are found

Coverage Types

1. **Statement Coverage** - Was this statement Executed?
2. **Path Coverage** - Was this code path executed?
3. **Decision/Branch Coverage** - Was every path of decision executed?
4. **Function Coverage** - Was Every Function is executed
5. **Synchronization Coverage** - Deals with multiple threads and Parallel Processing

Testing in Python

Several Testing Frameworks in Python:

- Unittest
- nose
- **pytest**

Why Pytest?

- Powerful features
- Less Boilerplate code
- Assertions are more natural

```
    assert == 'python' -> pytest

self.assertEquals('python') -> unittest
```

Pytest - Getting Started

- $ - pip install pytest
- Create a module to hold your test (e.g. test_cool_functions.py)
- Write tests inside the module
- Run tests by executing

```
$ - py.test [-v] test_cool_function.py
```

Pytest - Example

1. Code

```
def factorial(num):
    if num < 1:
        return 1
    else:
        return num * factorial(num - 1)
```

2. Write Unit tests

```
def test_factorial():
    assert pycon2017.factorial(4) == 24
```

3. Run the tests

```
$ pytest -v test_pycon2017.py
============================= test session starts =============================
platform win32 -- Python 3.6.0, pytest-3.0.5, py-1.4.32, pluggy-0.4.0 -- C:\Users\Haroon Rashid\Anaconda3\python.exe
cachedir: .cache
rootdir: C:\Users\Haroon Rashid\Desktop\Machine Learning\git\pycon2017, inifile:
collecting ... collected 1 items

test_pycon2017.py::test_factorial PASSED

========================== 1 passed in 0.03 seconds ===========================
```

Skip/Run Tests Selectively

- Selectively skip tests

```
@pytest.mark.skip(reason)
```

- Skip tests when a certain condition is met

```
@pytest.mark.skipif(condition, reason)
```

- Run all tests with a certain keyword

```
pytest -k keyword test_file.py
```

Skip/Run Tests Selectively - Continued

- Custom Markers
    - Mark tests to run on a certain operating system

```
@pytest.mark.windows

@pytest.mark.mac

pytest -m mac -v test_file.py
```

Pytest Fixtures

- Define Reusable components required by your tests
- Avoid setup and teardown modules
- Pytest combines fixtures to tests automatically

```
@pytest.fixture()
def fixture_object():
    return FixtureObject()

def test_new_method(fixture_object):
    assert fixture_object.is_fixture == True
```

https://docs.pytest.org/en/latest/builtin.html for more info on built in fixtures provided by **pytest**

Parameterization

- Allows to combine multiple tests into one

Code under Test

```
def sqaure(num):
    return num * num
```

No Parameterization ✖

```
def test_square_1():
    assert pycon2017.square(2) == 4


def test_square_2():
    assert pycon2017.square(5) == 25


def test_square_3():
    assert pycon2017.square(7) == 49
```

Parameterization ✔

```
@pytest.mark.parametrize('input, output', [(2, 4), (5, 25), (7, 49)])
def square(input, output):
    assert pycon2017.square(input) == output
```

Testing for Exceptions

- **pytest.raises** tests for exceptions

```
with pytest.raises(TypeError):
            ‘2’ + 2
```

- Running above code will make test pass because we are testing for exception

Test Automation - CI

- **Travis**, **AppVeyor** and **Jenkins** Continuous Integration
- Third Party service that builds (and run tests) every time code is pushed
- Free for Open Source Projects
- https://travis-ci.org/

Test Automation - Coveralls

- Third Party Service for keeping track of test coverage of project
- Free for open source Projects

Case Study - Stingray

https://github.com/StingraySoftware/stingray

ThankYou!!

@haroonrashid235

haroon.rashid235@gmail.com