Testing and TDD in Python

Núria Pujol
Laura Pérez

PyladiesBCN

May 14, 2015

www.meetup.com/PyLadies-BCN/

Pyladies

Pyladies is an **international mentorship group** with a focus on helping more women become active participants and leaders in the Python open-source community.

It comprises different groups around the world, including PyladiesBCN! Do you want to learn more about us?

- **On twitter:** @PyLadiesBCN
- **On meetup:** http://meetup.com/PyLadies-BCN
- **On mailing list:** pyladies-bcn@googlegroups.com

Testing our code

Testing is creating some tests that our code has to overcome.

- 'Extra' lines of code to guarantee that our program or functionality do exactly what it should do.
- Help us to detect software bugs that are difficult to detect using other techniques.
- Distribute them with our code is a good habit and software quality indicator.

When to write tests?

- Projects with many collaborators.
- Projects that has to evolve in different stages.
- Projects with different teams for different software parts.
- ...
- ALWAYS!!

Tests are the best project documentation for developers.

Testing utilities

There are different types of testing tools and libraries:

- **Unit testing tools.**
- Mock testing tools.
- Fuzz testing tools.
- Web testing tools.
- GUI testing tools.

https://wiki.python.org/moin/PythonTestingToolsTaxonomy

What is unit testing?

Test bench focused on catching bugs and detecting construction errors.

General rules:

- Every test is focused on verifying one tiny part or functionality (usually at function level or class level).
- Must be fully independent.
- Names must be descriptive and grouped in a logic way.
- Must reflect requirements accomplishments.

Unit testing Python Libraries

- **Unittest/Unittest2** (standard Python Library)
- **Py.test**
- Nose
- Doctest
- Others

Unittest test structure

```python
import unittest

class Test(unittest.TestCase):
    @classmethod
    def setUpClass(cls):
        pass

    def setUp(self):
        pass

    def test_(self):
        pass

    def tearDown(self):
        pass

    @classmethod
    def tearDownClass(cls):
        pass

if __name__ == '__main__':
    unittest.main()
```

Unittest test structure

```python
#test_myfunctions.py
import unittest
import myfunctions as f

class TestMyFunctions(unittest.TestCase):

    def test_len_string(self):
        self.assertEqual(f.len_string("Nuria"), 5)
        with self.assertRaises(TypeError):
            f.len_string(100)

    def test_higher_5(self):
        self.assertTrue(f.higher_than_5(100))
        self.assertFalse(f.higher_than_5(1))

if __name__ == '__main__':
    unittest.main()
```

Unittest test structure

When we run these tests:

```
python test_myfunctions.py
.F
======================================================================
FAIL: test_len_string (__main__.TestMyFunctions)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "test_myfunctions.py", line 9, in test_len_string
    self.assertEqual(f.len_string("Nuria"), 5)
AssertionError: 4 != 5

----------------------------------------------------------------------
Ran 2 tests in 0.000s
FAILED (failures=1)
```

Unittest test structure

Uups!! Some error ocurred!!
We have to try to solve that bug and run our tests again.

```
$ python test_myfunctions.py
..
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

Great!! Now everything is OK.

Exercices

Let’s practice!!!

Py.test test structure

```python
import pytest
import myfunctions as f

def test_len_string():
  assert f.len_string("Nuria") == 5
  with pytest.raises(TypeError):
    f.len_string(100)

def test_higher_5():
  assert f.higher_than_5(100) == True
  assert f.higher_than_5(1) == False
```

Py.test test structure

```
$ py.test test_myfunctions.py
============== test session starts ===============
platform linux2 -- Python 2.7.6 -- py-1.4.26 -- pytest-2.7.0
rootdir: /home/nuria/PyLadies/Testing/testing_and_TDD, inifile:
collected 2 items

test_myfunctions.py ..

============ 2 passed in 0.03 seconds ============
```

Doctest test structure

```python
def higher_than_5(input):
  """
  >>> higher_than_5(100)
  True
  >>> higher_than_5(1)
  False

  """
  if input > 5:
    return True

  return False

if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

Doctest test structure

```
$ python -m doctest -v myfunctions.py
Trying:
    higher_than_5(100)
Expecting:
    True
ok
Trying:
    higher_than_5(1)
Expecting:
    False
ok
1 items passed all tests:
   2 tests in lines_code.higher_than_5
2 tests in 2 items.
2 passed and 0 failed.
Test passed.
```

Pros and cons of testing

PROS:

- Ensure stability of our code.
- Increase our productivity.
- Facilitate developing code in a team.

CONS:

- We have to write extra code.
- Writing good tests is not simple.
- If you don't know how to use it, don't use it!

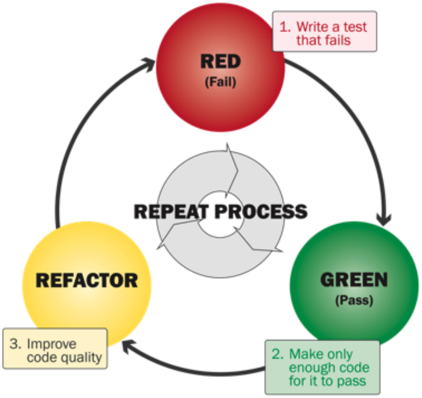
TDD (Test Driven Development)

TDD is a development technique related to Agile methodologies

- Development is divided in simple tasks (BABY STEPS).
- First we develop our tests and then our code.
- It's a cycling process.

- Ensure that your code until current passing test is ok.
- Easier to estimate deadlines.

TDD Cycle



RED
(Fail)
1. Write a test that fails
GREEN
(Pass)
2. Make only enough code for it to pass
REFACTOR
3. Improve code quality
REPEAT PROCESS

Exercices

Now we are going to take practice with some exercises.

ARE YOU READY?

Thank you :) @PyLadiesBCN