

SEES: Test Driven Development

FOSSEE

February 10, 2015

Outline

- 1 Introduction
- 2 First Test
- 3 Python Testing Frameworks
 - nose

Outline

1 Introduction

2 First Test

3 Python Testing Frameworks

- nose

Objectives

At the end of this section, you will be able to:

- Write your code using the TDD paradigm.
- Use the nose module to test your code.

What is TDD?

The basic steps of TDD are roughly as follows –

- 1 Decide upon feature to implement and methodology of testing
- 2 Write tests for feature decided upon
- 3 Just write enough code, so that the test can be run, but it fails.
- 4 Improve the code, to just pass the test and at the same time passing all previous tests.
- 5 Run the tests to see, that all of them run successfully.
- 6 Refactor the code you've just written – optimize the algorithm, remove duplication, add documentation, etc.
- 7 Run the tests again, to see that all the tests still pass.
- 8 Go back to 1.

What is TDD?

The basic steps of TDD are roughly as follows –

- 1 Decide upon feature to implement and methodology of testing
- 2 Write tests for feature decided upon
- 3 Just write enough code, so that the test can be run, but it fails.
- 4 Improve the code, to just pass the test and at the same time passing all previous tests.
- 5 Run the tests to see, that all of them run successfully.
- 6 Refactor the code you've just written – optimize the algorithm, remove duplication, add documentation, etc.
- 7 Run the tests again, to see that all the tests still pass.
- 8 Go back to 1.

What is TDD?

The basic steps of TDD are roughly as follows –

- 1 Decide upon feature to implement and methodology of testing
- 2 Write tests for feature decided upon
- 3 Just write enough code, so that the test can be run, but it fails.
- 4 Improve the code, to just pass the test and at the same time passing all previous tests.
- 5 Run the tests to see, that all of them run successfully.
- 6 Refactor the code you've just written – optimize the algorithm, remove duplication, add documentation, etc.
- 7 Run the tests again, to see that all the tests still pass.
- 8 Go back to 1.

What is TDD?

The basic steps of TDD are roughly as follows –

- 1 Decide upon feature to implement and methodology of testing
- 2 Write tests for feature decided upon
- 3 Just write enough code, so that the test can be run, but it fails.
- 4 Improve the code, to just pass the test and at the same time passing all previous tests.
- 5 Run the tests to see, that all of them run successfully.
- 6 Refactor the code you've just written – optimize the algorithm, remove duplication, add documentation, etc.
- 7 Run the tests again, to see that all the tests still pass.
- 8 Go back to 1.

What is TDD?

The basic steps of TDD are roughly as follows –

- 1 Decide upon feature to implement and methodology of testing
- 2 Write tests for feature decided upon
- 3 Just write enough code, so that the test can be run, but it fails.
- 4 Improve the code, to just pass the test and at the same time passing all previous tests.
- 5 Run the tests to see, that all of them run successfully.
- 6 Refactor the code you've just written – optimize the algorithm, remove duplication, add documentation, etc.
- 7 Run the tests again, to see that all the tests still pass.
- 8 Go back to 1.

What is TDD?

The basic steps of TDD are roughly as follows –

- 1 Decide upon feature to implement and methodology of testing
- 2 Write tests for feature decided upon
- 3 Just write enough code, so that the test can be run, but it fails.
- 4 Improve the code, to just pass the test and at the same time passing all previous tests.
- 5 Run the tests to see, that all of them run successfully.
- 6 Refactor the code you've just written – optimize the algorithm, remove duplication, add documentation, etc.
- 7 Run the tests again, to see that all the tests still pass.
- 8 Go back to 1.

What is TDD?

The basic steps of TDD are roughly as follows –

- 1 Decide upon feature to implement and methodology of testing
- 2 Write tests for feature decided upon
- 3 Just write enough code, so that the test can be run, but it fails.
- 4 Improve the code, to just pass the test and at the same time passing all previous tests.
- 5 Run the tests to see, that all of them run successfully.
- 6 Refactor the code you've just written – optimize the algorithm, remove duplication, add documentation, etc.
- 7 Run the tests again, to see that all the tests still pass.
- 8 Go back to 1.

What is TDD?

The basic steps of TDD are roughly as follows –

- 1 Decide upon feature to implement and methodology of testing
- 2 Write tests for feature decided upon
- 3 Just write enough code, so that the test can be run, but it fails.
- 4 Improve the code, to just pass the test and at the same time passing all previous tests.
- 5 Run the tests to see, that all of them run successfully.
- 6 Refactor the code you've just written – optimize the algorithm, remove duplication, add documentation, etc.
- 7 Run the tests again, to see that all the tests still pass.
- 8 Go back to 1.

Outline

1 Introduction

2 First Test

3 Python Testing Frameworks

- nose

First Test – GCD

- simple program – GCD of two numbers
- What are our code units?
 - Only one function `gcd`
 - Takes two numbers as arguments
 - Returns one number, which is their GCD

```
c = gcd(44, 23)
```

- `c` will contain the GCD of the two numbers.

Test Cases

- Important to have test cases and expected outputs even before writing the first test!
- $a = 48, b = 48, GCD = 48$
- $a = 44, b = 19, GCD = 1$
- Tests are just a series of assertions
- True or False, depending on expected and actual behavior

Test Cases – general idea

```
tc1 = gcd(48, 64)
if tc1 != 16:
    print "Failed for a=48, b=64. Expected 16. \
    Obtained %d instead." % tc1
    exit(1)
```

```
tc2 = gcd(44, 19)
if tc2 != 1:
    print "Failed for a=44, b=19. Expected 1. \
    Obtained %d instead." % tc2
    exit(1)
```

```
print "All tests passed!"
```

- The function gcd doesn't even exist!

Test Cases – code

- Let us make it a function!
- Use assert!

Test Cases – code

```
# gcd.py  
def test_gcd():  
    assert gcd(48, 64) == 16  
    assert gcd(44, 19) == 1  
  
test_gcd()
```

Stubs

- First write a very minimal definition of `gcd`

```
def gcd(a, b):  
    pass
```

- Written just, so that the tests can run
- Obviously, the tests are going to fail

gcd.py

```
def gcd(a, b):  
    pass  
  
def test_gcd():  
    assert gcd(48, 64) == 16  
    assert gcd(44, 19) == 1  
  
if __name__ == '__main__':  
    test_gcd()
```

First run

```
$ python gcd.py
Traceback (most recent call last):
  File "gcd.py", line 9, in <module>
    test_gcd()
  File "gcd.py", line 5, in test_gcd
    assert gcd(48, 64) == 16
AssertionError
```

- We have our code unit stub, and a failing test.
- The next step is to write code, so that the test just passes.

Euclidean Algorithm

- Modify the `gcd` stub function
- Then, run the script to see if the tests pass.

```
def gcd(a, b):  
    if a == 0:  
        return b  
    while b != 0:  
        if a > b:  
            a = a - b  
        else:  
            b = b - a  
    return a
```

```
$ python gcd.py  
All tests passed!
```

• Success!

Euclidean Algorithm – Modulo

- Repeated subtraction can be replaced by a modulo
- modulo of $a \% b$ is always less than b
- when $a < b$, $a \% b$ equals a
- Combine these two observations, and modify the code

```
def gcd(a, b):  
    while b != 0:  
        a, b = b, a % b  
    return a
```

- Check that the tests pass again

Euclidean Algorithm – Recursive

- Final improvement – make `gcd` recursive
- More readable and easier to understand

```
def gcd(a, b):  
    if b == 0:  
        return a  
    return gcd(b, a%b)
```

- Check that the tests pass again

Document gcd

- Undocumented function is as good as unusable
- Let's add a docstring & We have our first test!

```
def gcd(a, b):  
    """Returns the Greatest Common Divisor of the  
    two integers passed as arguments.  
  
    Args:  
        a: an integer  
        b: another integer  
  
    Returns: Greatest Common Divisor of a and b  
    """  
    if b == 0:  
        return a  
    return gcd(b, a%b)
```

Persistent Test Cases

- Tests should be pre-determined and written, before the code
- The file shall have multiple lines of test data
- Separates the code from the tests

Separate test_gcd.py

```
from gcd import gcd

def test_gcd():
    assert gcd(48, 64) == 16
    assert gcd(44, 19) == 1

if __name__ == '__main__':
    test_gcd()
```

Outline

1 Introduction

2 First Test

3 Python Testing Frameworks

- nose

Python Testing Frameworks

- Testing frameworks essentially, ease the job of the user
- Python provides two frameworks for testing code
 - `unittest` framework
 - `doctest` module
- `nose` is a package to help test

nose tests

- It is not easy to organize, choose and run tests scattered across multiple files.
- `nose` module aggregate these tests automatically
- Can aggregate `unittests` and `doctests`
- Allows us to pick and choose which tests to run
- Helps output the test-results and aggregate them in various formats
- Not part of the Python distribution itself

```
$ apt-get install python-nose
```

- Run the following command in the top level directory

```
$ nosetests
```