
PYTHON UNIT TESTING

Brett Langdon | @brett_langdon | <http://brett.is/>



Who am I?

- * Brett Langdon
- * Software Engineer - Magnetic
- * brett@blangdon.com
- * <http://brett.is>
- * [@brett_langdon](https://twitter.com/brett_langdon)
- * github.com/brettlangdon

Summary

- * What is Unit Testing

- * *unittest*

- * *doctest*

- * Frameworks

- * Mocking

- * Dependency Injection

Code Samples/Slides

* <http://www.github.com/PythonBuffalo/unit-testing>

Unit Testing

- * Testing individual units of code rather than the code as a whole
- * Test each class/method/function separately from all other code
 - * Code Isolation
- * TDD - Test Driven Development

unittest

- * Built-in Python library (since 2.1)
- * “PyUnit”
- * **test fixtures** - preparation needed to perform tests
- * **test case** - a test for the smallest unit of testing
- * **test suite** - collection of test cases or suites to be run together
- * **test runner** - component used to gather and run test suites

unittest - Test Case

```
import unittest

class SimpleTest(unittest.TestCase):
    def test_method(self):
        assert 1 == 1, 'How is 1 not equal to 1'
        assert(1 == 2, 'do not ever do this')    # Passes

        self.assertTrue(1 == 1, 'How is 1 not equal to 1')
        self.assertEqual(1, 1, 'How is 1 not equal to 1')
```

unittest - Running

```
$ cat sample_test.py
```

```
import unittest
```

```
class SampleTest(unittest.TestCase):
```

```
...
```

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

```
$ python sample_test.py
```

```
$ python -m unittest sample_test
```


unittest - Exceptions

```
import unittest
```

```
def add_two(number):  
    return number + 2
```

```
class TestExceptions(unittest.TestCase):
```

```
    def test_add_two(self):  
        self.assertEqual(4, add_two(2))  
        self.assertRaises(TypeError, add_two, 'two')
```

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

unittest - Assertions

- * **assertEqual**(*first, second, msg=None*)
- * **assertNotEqual**(*first, second, msg=None*)
- * **assertTrue**(*expr, msg=None*)
- * **assertFalse**(*expr, msg=None*)
- * **assertIs**(*first, second, msg=None*)
- * **assertIsNot**(*first, second, msg=None*)
- * **assertIsNone**(*expr, msg=None*)
- * **assertIsNotNone**(*expr, msg=None*)
- * **assertIn**(*first, second, msg=None*)
- * **assertNotIn**(*first, second, msg=None*)
- * **assertIsInstance**(*obj, cls, msg=None*)
- * **assertNotIsInstance**(*obj, cls, msg=None*)
- * **assertRaises**(*exception, callable, *args, **kwds*)
- * **assertRaises**(*exception*)
- * **assertGreater**(*first, second, msg=None*)
- * **assertLess**(*first, second, msg=None*)
- * **assertRaisesRegexp**(*exception, regexp*)
- * ... and **more**

doctest

- * Documentation defined unit tests
- * Written as docstrings in scripts
- * Looks like python interactive shell

doctest - Definition

```
def factorial(num):  
    """A function to compute the factorial of a number  
  
    >>> factorial(5)  
    120  
    >>> factorial(-5)  
    1  
    """  
    if num <= 0:  
        return 1  
    else:  
        return num * factorial(num - 1)
```


doctest - Running

```
$ cat my_test.py
```

```
def factorial(num):  
    """
```

```
>>> factorial(5)  
120  
    """
```

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

```
$ python my_test.py
```

```
$ python -m doctest my_test.py
```

doctest - Verbose

```
$ python my_test.py -v
```

```
Trying:
```

```
    factorial(5)
```

```
Expecting:
```

```
    120
```

```
ok
```

```
Trying:
```

```
    factorial(25)
```

```
Expecting:
```

```
15511210043330985984000000L
```

```
ok
```

```
Trying:
```

```
    factorial(-5)
```

```
Expecting:
```

```
    1
```

```
ok
```

```
1 items had no tests:
```

```
    __main__
```

```
1 items passed all tests:
```

```
    3 tests in
```

```
        __main__.factorial
```

```
3 tests in 2 items.
```

```
3 passed and 0 failed.
```

```
Test passed.
```


doctest - Exceptions

```
def remove_newline(text):
```

```
    """Remove all new line characters from provided text
```

```
>>> remove_newline('hello\\nworld')  
'helloworld'
```

```
>>> remove_newline(5)
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
  File "<stdin>", line 2, in remove_newline
```

```
AttributeError: 'int' object has no attribute 'replace'  
"""
```

```
return text.replace('\\n', '')
```

doctest - Variables

```
def remove_twos(numbers):  
    """Function to remove all 2's from a list  
  
    >>> numbers = [3, 4, 8, 2, 4, 2, 3, 5, 2]  
    >>> results = remove_twos(numbers)  
    >>> type(results)  
    <type 'generator'>  
    >>> list(results)  
    [3, 4, 8, 4, 3, 5]  
    """  
  
    for num in numbers:  
        if num != 2:  
            yield num
```


doctest - Classes

```
class TestClass(object):
    """
    >>> tc = TestClass([5, 6, 6, 6])
    >>> tc.add_number(2)
    >>> tc.add_numbers([5, 6, 7])
    >>> tc.numbers
    set([2, 5, 6, 7])
    """

    def __init__(self, numbers):
        """
        >>> tc = TestClass([3, 5, 5])
        >>> tc.numbers
        set([3, 5])
        """
        self.numbers = set(numbers)
```

```
def add_number(self, number):
    """
    >>> tc = TestClass([5])
    >>> tc.add_number(5)
    >>> tc.numbers
    set([5])
    """
    self.numbers.add(number)

def add_numbers(self, numbers):
    """
    >>> tc = TestClass([5, 6, 6])
    >>> tc.add_numbers([5, 6, 7])
    >>> tc.numbers
    set([5, 6, 7])
    """
    self.numbers |= set(numbers)
```

Frameworks

- * Third Party Testing Framework Modules

- * **Py.Test** - <http://pytest.org/>

- * **Nose** - <https://nose.readthedocs.org/>

- * **PyUnit** (kind of old now)- <http://pyunit.sourceforge.net/>

Mock

- * Third Party Module for mocking/patching
- * Python 2.4+
- * <https://pypi.python.org/pypi/mock>
- * <http://www.voidspace.org.uk/python/mock/>

Mock - Mock Object

```
import mock

mocked = mock.Mock()
print mocked
# <Mock id='4459839248'>

print mocked()
# <Mock name='mock()'
id='4459873552'>

mocked.return_value = 5
print mocked()
# 5
```

```
def dummy():
    return 6

mocked.side_effect = dummy
print mocked()
# 6

print mocked.attribute
# <Mock
name='mock.attribute'
id='4459836432'>
```


Mock - Assertions

```
import mock
```

```
mocked_obj = mock.Mock()  
mocked_obj()  
mocked_obj.called    # True  
mocked_obj.reset_mock()  
mocked_obj.called    # False
```

```
mocked_obj('some', 'args')  
mocked_obj.assert_called_with('some', 'args')  
mocked_obj.call_args    # call('some', 'args')  
mocked_obj.assert_called_with('other', 'arguments')  
# raises AssertionError
```

```
mocked_obj('other', 'arguments')  
mocked_obj.call_count    # 2
```

Mock - Methods/Properties

- * **assert_called_with**(*args, **kwargs)
- * **assert_called_once_with**(*args, **kwargs)
- * **assert_any_call**(*args, **kwargs)
- * **assert_has_calls**(calls, any_order=False)
- * **reset_mock**()
- * **mock_add_spec**(spec, spec_set=False)
- * **attach_mock**(mock, attribute)
- * **configure_mock**(**kwargs)
- * **called**
- * **call_count**
- * **return_value**
- * **side_effect**
- * **call_args**
- * **call_args_list**
- * **method_calls**
- * **mock_calls**

Mock - Mocks

- * `class Mock(spec=None, side_effect=None, return_value=DEFAULT, wraps=None, name=None, spec_set=None, **kwargs)`
- * `class MagicMock(*args, **kw)`
- * `class PropertyMock(*args, **kwargs)`
- * `class NonCallableMock(spec=None, wraps=None, name=None, spec_set=None, **kwargs)`
- * `class NonCallableMagicMock(*args, **kw)`

Mock - Patching

```
import mock
```

```
with mock.patch('urllib.quote_plus', mock.Mock(return_value='something')) :  
    print uses_quote_plus('http://pythonbuffalo.org/')  
    # something
```

```
with mock.patch('urllib.quote_plus') as mocked_quote_plus:  
    mocked_quote_plus.return_value = 'http://github.com/'  
    print uses_quote_plus('http://www.brett.is/')  
    # http://github.com/
```


Mock - Patching Dicts

```
import mock

data = {}
with mock.patch.dict(data, {'some': 'values'}):
    assert 'some' in data
    assert data['some'] == 'values'

# no longer patched, should be the {}
assert 'some' not in data
```

Mock - Patching Objects

```
import mock
import random

with mock.patch.object(random, "random", return_value=5):
    print random.random()
# 5
```


Mock - Patching Decorator

```
import mock

def func(url):
    return do_something(url)

@mock.patch('__main__.func', mock.Mock(return_value='http://github.com/'))
def test(url):
    return func(url)

print test('http://pythonbuffalo.org/')
# http://github.com/
```

Dependency Injection

```
class DBCaller(object):  
    def __init__(self):  
        self.connection = DBConnection()  
  
    def get_random(self):  
        results = self.connection.fetch_all_the_things()  
        for result in results:  
            if random.random() > .5:  
                yield result  
  
    def get_time(self):  
        self.connection.execute('get where time < %s',  
                                (time.time(), ))  
        return self.connection.fetch_all()
```


DI - Patching

```
import mock

class DBCaller(object):
    ...

with mock.patch("db_connection") as db:
    with mock.patch("random.random", return_value=1):
        with mock.patch("time.time", return_value=1371504225):
            expected = [...]
            caller = DBCaller()
            caller.fetch_all_the_things.return_value = expected
            results = list(caller.get_random())
            assert results == expected
            results = caller.get_time()
```

Dependency Injection, Cont.

```
import mock
```

```
class DBCaller(object):
```

```
    def __init__(self, connection=DBConnection):  
        self.connection = connection()
```

```
    def get_random(self, random=random.random):  
        results = self.connection.fetch_all_the_things()  
        for result in results:  
            if random() > .5:  
                yield result
```

```
    def get_time(self, time=time.time):  
        self.connection.execute('get where time < %s',  
                                (time(), ))  
        return self.connection.fetch_all()
```


DI - Mocking

```
import mock
```

```
class DBCaller(object):
```

```
    ...
```

```
expected = [...]
```

```
mock_db = mock.Mock(spec=DBConnection)
```

```
mock_random = mock.Mock(return_value=1)
```

```
mock_time = mock.Mock(return_vaue=1371504225)
```

```
caller = DBCaller(connection=mock_db)
```

```
caller.fetch_all_the_things.return_value = expected
```

```
results = list(caller.get_random(random=mock_random))
```

```
assert results == expected
```

```
results = caller.get_rime(time=mock_time)
```

FIN
