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
* github.com/brettlangdon

Summary

* What is Unit Testing
* *unittest*
* *doctest*
* Frameworks
* Mocking
* Dependency Injection

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')
```

unitest - 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

Dependency Injection

FIN