



Unit Testing in Python

James Brucker

Python Testing Frameworks

We will cover these two:

- **unittest** - part of the Python library, similar to JUnit 3
- **DocTest** - test by example, part of the Python library

Other testing frameworks:

- **Py.Test** - very simple "assert" syntax.
 - can also run unittest style tests
- **Nose**
- <https://wiki.python.org/moin/PythonTestingToolsTaxonomy>

unittest simple example

```
import unittest
```

class extends TestCase

```
class TestBuiltins(unittest.TestCase):
    """Test some python built-in methods"""
    def test_len(self):
        self.assertEqual(5, len("hello"))
        self.assertEqual(3, len(['a', 'b', 'c']))
        # edge case
        self.assertEqual(0, len(""))

    def test_str_upper(self):
        self.assertTrue("ABC".isupper())
        self.assertFalse("ABc".isupper())
        s = "" # edge case
        self.assertFalse(s.isupper())
```

Run tests from the command line

Run all tests or just specific test.

```
cmd> python -m unittest test_module1 test_module2
```

```
cmd> python -m unittest module.TestClass
```

```
cmd> python -m tests/test_module.py
```

Other Ways to Run tests

1. Let the IDE run them for you.
2. Use a separate test script or build tool.
3. Add a "main" script to end of your Test class...

```
import unittest

class TestBuiltins(unittest.TestCase):
    """Test some python built-in method"""
    def test_len(self):
        self.assertEqual(5, len("hello"))
        self.assertEqual(3, len(['a', 'b', 'c']))

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

What Can You assert?

```
assertTrue( gcd(-3,-5) > 0 )  
assertFalse( stack.isFull() )  
assertEqual( 2*2, 4)  
assertNotEqual( "a", "b")  
assertIs(a, b)                # test "a is b"  
assertIsNot(a, b)             # test "a is not b"  
assertIn( a, list)            # test "a in list"  
assertIsInstance(3, int)      # test isinstance(a,b)  
assertListEqual( list1, list2 )
```

Many more!

See "unittest" in the Python Library docs.

Skip a Test or Fail a Test

```
import unittest

class MyTest(unittest.TestCase):

    @unittest.skip("Not done yet")

    def test_read_file(self):
        pass

    def test_fraction_constructor(self):
        self.fail("Write this test!")
```

Test for Exception

What if your code should throw an exception?

```
def average( list ):
    """Compute average of a list of numbers.
    The list must not be empty.
    """
    if len(list) == 0:
        raise AttributeError("List is empty")
    return sum(list)/len(list)
```


Test for Exception

`assertRaises` expects a block of code to raise an exception:

```
def test_empty_list_throws_exception(self):  
    testlist = []  
    with self.assertRaises(AttributeError):  
        x = average(testlist)
```

A Stack Example

- ❑ A **Stack** implements common stack data structure.
- ❑ You can `push()`, `pop()`, and `peek()` elements.
- ❑ Throws **StackException** if you do something stupid.

Stack
+ Stack(capacity)
+ capacity(): int
+ size(): int
+ isEmpty(): boolean
+ isFull(): boolean
+ push(T): void
+ pop(): T
+ peek(): T

Use setUp() to create test fixture

setUp() is called **before each test**.

```
import unittest

class StackTest(unittest.TestCase):
    """Create a new test fixture before each test"""
    def setUp(self):
        self.capacity = 5
        self.stack = Stack(capacity)

    def test_new_stack_is_empty(self):
        self.assertTrue( self.stack.isEmpty() )
        self.assertFalse( self.stack.isFull() )
        self.assertEqual( 0, self.stack.size() )
```

Test for Exceptions

```
import unittest

class StackTest(unittest.TestCase):
    """Create a test fixture for the tests"""
    def setUp(self):
        self.capacity = 5
        self.stack = Stack(capacity)

    def test_pop_empty_stack(self):
        """stack.pop() should throw exception"""
        with self.assertRaises(StackException):
            self.stack.pop()
```

Doctest

Include runnable code inside Python DocStrings.

Provides **example** of how to use code and executable tests!

```
def add(a,b) :  
    """Compute the sum of two values.  
  
    >>> add(3,4)  
    7  
    >>> add('a','b')  
    'ab'  
    """  
    return a+b
```

Running Doctest

Here is one way...

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

Failed: 0, Attempted: 2

Testing is Not So Easy!

These examples are *trivial* tests to show the syntax.

Real tests are much more thoughtful and demanding.

Designing good tests forces you to *think* about what the code should do, and what may go wrong.

Good tests are often very short... but many of them.

References

Python Official Docs (easy to read, has examples)

`https://docs.python.org/3/library/unittest.html`

Python Hitchhiker's Guide to Testing

`https://docs.python-guide.org/writing/tests/`

- Examples of many common test tools

Python Cookbook, Chapter 14

How to test many common situations, including I/O