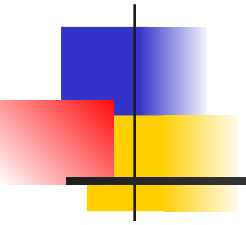

Unit Testing in Python

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
- Mock objects - create "fake" external components
- https://wiki.python.org/moin/PythonTestingToolsTaxonomy

unittest simple example

```
import unittest

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() )
```

class extends `TestCase`

Run tests from the command line

Run all tests or just specific test.

```
cmd> python -m unittest test_module

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 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()
```

Demo: Try it Yourself

Test math.sqrt() and math.pow().

```
import unittest
import math

class TestMath(unittest.TestCase):
    def test_sqrt(self):
        self.assertEqual(5, math.sqrt(25))
        self.assertEqual(0, math.sqrt(0))

    def test_pow(self):
        #TODO Write some tests of math.pow(x,n)
```

Demo: Run Your Tests

Run on the command line:

```
cmd>  python -m unittest math_test.py
..
----------------------------------------------------------------
Ran 2 tests in 0.000s
```

Run with verbose (-v) output

```
cmd>  python -m unittest -v math_test.py
test_sqrt (math_test.MathTest) ... ok
test_pow (math_test.MathTest) ... ok
----------------------------------------------------------------
Ran 2 tests in 0.000s
```

Demo: Write a Failing Test

Test math.sqrt() with a negative argument

```
import unittest
import math

class TestMath(unittest.TestCase):
    def test_sqrt_of_negative(self):
        self.assertEqual(4, math.sqrt(-16))
```

Demo: Run the Failing Test

Run on the command line:

```
cmd>  python -m unittest math_test.py
..E
======================================================================
ERROR: test_sqrt_negative (math_test.MathTest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "math_test.py", line 10, in test_sqrt_negative
    self.assertEqual(4, math.sqrt(-16))
ValueError: math domain error


----------------------------------------------------------------------
Ran 3 tests in 0.001s
FAILED (errors=1)
```

Tests Outcomes

success: passes all "assert"

failure: fails an "assert" but code runs OK

error: test caused an error, such as exception raised

What Can You assert?

```
assertTrue( gcd(-3,-5) > 0 )
assertFalse( "hello".isupper() )
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 ) # all elments equal
```

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_add_fractions(self):
        pass

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

Test for Exception

What if your code should throw an exception?

```
def test_sqrt_of_negative( self ):
    """sqrt of a negative number should throw
       ValueError.
    """
    self.assert????( math.sqrt(-1) )
```

Test for Exception

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

```
def test_sqrt_of_negative(self):
    with self.assertRaises(ValueError):
        math.sqrt(-1)
```

Test involving Floating Point

Calculations using floating point values often result in rounding error or finite-precision error.

This is normal.

If you need to test a function with rounding error, use `assertAlmostEqual`

```
def test_with_limited_precision( self ):
    self.assertAlmostEqual(
        1.0, math.sin(math.pi/2), places=8 )
```

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 Stack Exception

```
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

Run doctest in code:

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

```
Failed: 0, Attempted: 2
```

Or run doctest using command line (no '__main__'):

```
cmd> python -m doctest -v file1.py
2 tests in 5 items.
2 passed and 0 failed.
Test passed.
```

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 makes you 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 testing tools

Python Cookbook, Chapter 14

How to test many common situations, including I/O