Unit Testing

Python `UnitTest` Library

Unit Testing is concerned with testing small chunks of a program, for example, testing a single class or a single method. Python has a library for unit testing called `unittest`. It provides several tools for creating and running unit tests.

One of the most important classes in `unittest` is called `TestCase` which provides a set of methods to compare values, set up tests and clean up after tests are finished. To write unit tests, we create a subclass of `TestCase` and write methods to do the actual testing. Typically we start all of these methods with the name test. Here's an example:

```
import unittest
class CheckNumbers(unittest.TestCase):
    def test_int_float(self):
        self.assertEqual(1, 1.0)
    # END test_int_float
# END CheckNumbers
```

If the assertion is found to be true, it returns ".", and if it fails, it returns "F".

Assetion Methods

Methods	Description
`assertEqual` `assertNotEqual`	Accept two comparable objects and ensure the named equality holds.
`assertTrue` `assertFalse`	Accept a single expression, and fail if the expression does not pass an IF test.

`assertGreater`
`assertGreaterEqual`
`assertLess`
`assertLessEqual`
	Accept two comparable objects and ensure the named inequality holds.

`asssertIn` `assertNotIn`	Ensure an element is (or is not) an element in a container object.
`assertIsNone` `assertIsNotNone`	Ensure an element is (or is not) the exact value None (but not another false value).
`assertSameElements`	Ensure two container objects have the same elements, ignoring the order.

`assertSequenceEqual`
`assertDictEqual`
`assertSetEqual`
`assertListEqual`
`assertTupleEqual`
	Ensure two containers have the same elements in the same order. If there's a failure, show a code diff comparing the two lists to see where they differ. The last four methods also test the type of the list.

`assertRaises`	Ensure a specific function call raises a specific exception.

To pass the `assertFalse` method the test should return False, None, 0, or an empty list, dictionary, string, set, or tuple. To pass the `assertTrue` method the test should return True, non-zero numbers, containers with values in.

Unit Testing

The `assertRaises` Method

Let's look at an example of the `assertRaises` method:

```
import unittest

def MyAverage(seq):
  return sum(seq) / len(seq)
# END average

class TestAverage(unittest.TestCase):
    def setUp(self):
        self.stats = StatsList([1,2,2,3,3,4])
    # END setUp

    def test_mean(self):
        self.assertEqual(self.stats.mean(), 2.5)
    # END test_mean

    def test_MyAverage(self):
      self.assertRaises(ZeroDivisionError, MyAverage, [])
    # END test_zero
# END CLASS TestAverage
```

The `setUp` is called individually before each test, so each test starts with a clean slate.

Built-in Exceptions

Some of the Python built-in exceptions are as follows:

Methods	Description
`AssertionError`	Raised when an `assert` statement fails.
`AttributeError`	Raised when an attribute reference or assignment fails.
`FloatingPointError`	Raised when a floating point operation fails.
`IOError`	Raised when an I/O operation (such as a `print` statement) fails for an I/O-related reason.
`IndexError`	Raised when a sequence subscript is out of range.
`MemoryError`	Raised when an operation runs out of memory but the situation may still be rescued by deleting some objects.
`OverflowError`	Raised when the result of an arithmetic operation is too large to be represented.
`RuntimeError`	Raised when an error is detected that doesn't fall in any of the other categories.
`SyntaxError`	Raised when the parser encounters a syntax error.
`ZeroDivisionError`	Raised when the second argument of a division or modulo operation is zero.

Test cases should never have side effects.