Unit Testing in Python

Md. Mohsin Uddin

East West University

mmuddin@ewubd.edu

July 31, 2021

unittest — Unit Testing Framework in Python

The **unittest** unit testing framework supports:

- test automation
- sharing of setup and shutdown code for tests
- aggregation of tests into collections
- independence of the tests from the reporting framework

unittest — Unit Testing Framework in Python

To achieve this, unittest supports some important concepts in an object-oriented way:

- **test fixture** - A test fixture represents the preparation needed to perform one or more tests, and any associated cleanup actions. This may involve, for example, creating temporary or proxy databases, directories, or starting a server process.
- **test case** - A test case is the individual unit of testing. It checks for a specific response to a particular set of inputs. unittest provides a base class, **TestCase**, which may be used to create new test cases.

Basic Example

The unittest module provides a rich set of tools for constructing and running tests.

```
import unittest
class TestStringMethods(unittest.TestCase):
    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')
    def test_isupper(self):
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())
    def test_split(self):
        s = 'hello world'
        self.assertEqual(s.split(), ['hello', 'world'])
        # check that s.split fails when the
        # separator is not a string
        with self.assertRaises(TypeError):
            s.split(2)

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

Command-Line Interface

The unittest module can be used from the command line to run tests from modules, classes or even individual test methods:

```
python -m unittest test_module1 test_module2
python -m unittest test_module.TestClass
python -m unittest test_module.TestClass.test_method
```

Command-Line Interface

```
python -m unittest tests/test_something.py
python -m unittest -v test_module
```

- Test modules can be specified by file path.
- Tests can be run with more detail (higher verbosity) by passing in the -v flag.

Organizing Test Code: setUp and tearDown

```
import unittest
class WidgetTestCase(unittest.TestCase):
    def setUp(self):
        self.widget = Widget('The widget')
    def test_default_widget_size(self):
        self.assertEqual(self.widget.size(), (50,50),
                         'incorrect default size')
    def test_widget_resize(self):
        self.widget.resize(100,150)
        self.assertEqual(self.widget.size(), (100,150),
                         'wrong size after resize')
    def tearDown(self):
        self.widget.dispose()
```

- The testing framework will automatically call **setUp()** method for every single test we run.
- If setUp() succeeded, **tearDown()** will be run whether the test method succeeded or not.

Organizing Test Code: setUpClass and tearDownClass

```
import unittest
class Test(unittest.TestCase):
    @classmethod
    def setUpClass(cls):
        cls._connection = createExpensiveConnectionObject()
    @classmethod
    def tearDownClass(cls):
        cls._connection.destroy()
        self.widget.dispose()
```

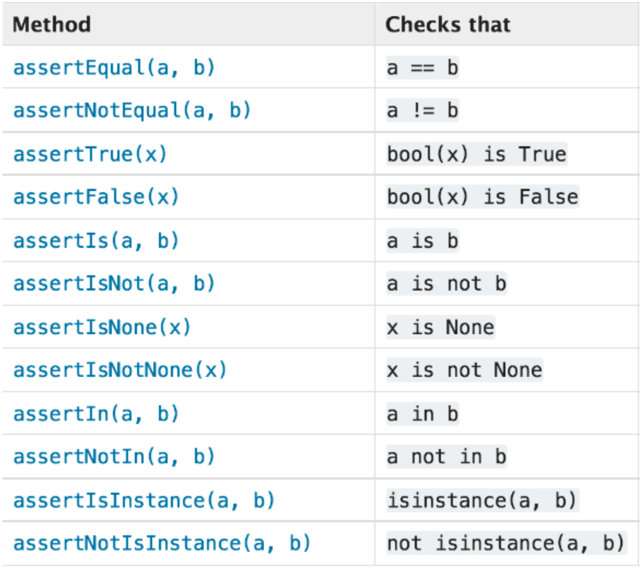
Skipping Tests and Expected Failures

```
class MyTestCase(unittest.TestCase):
    @unittest.skip("demonstrating␣skipping")
    def test_nothing(self):
        self.fail("shouldn't␣happen")
    def test_format(self):
        # Tests that work for only a certain version of the libr
        pass
    @unittest.skipUnless(sys.platform.startswith("win"), "")
    def test_windows_support(self):
        # windows specific testing code
        pass
    def test_maybe_skipped(self):
        if not external_resource_available():
            self.skipTest("external␣resource␣not␣available")
        # test code that depends on the external resource
        pass
```

Skipping Test Class

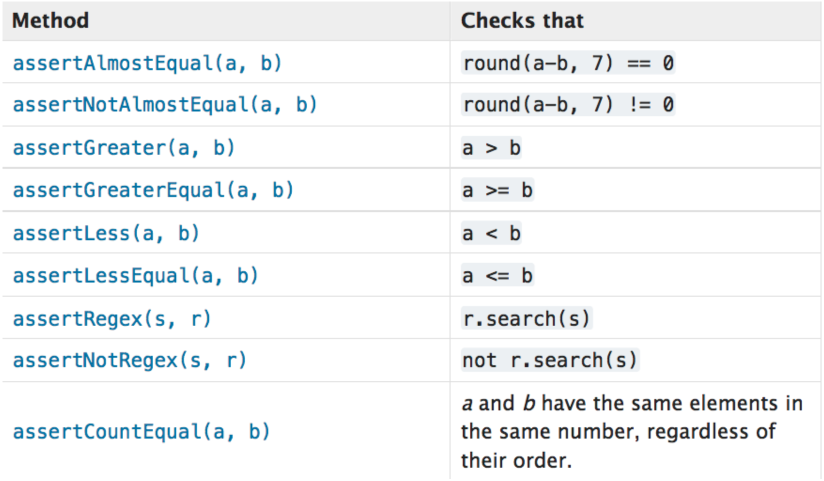
```
@unittest.skip("showing class skipping")
class MySkippedTestCase(unittest.TestCase):
    def test_not_run(self):
        pass
```

Assert Methods



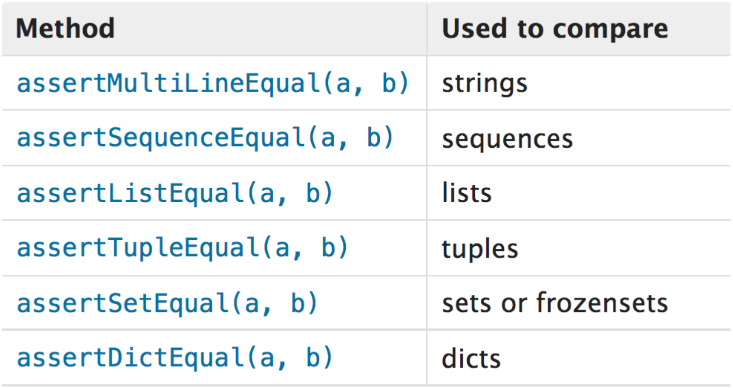
Method	Checks that
`assertEqual(a, b)`	`a == b`
`assertNotEqual(a, b)`	`a != b`
`assertTrue(x)`	`bool(x) is True`
`assertFalse(x)`	`bool(x) is False`
`assertIs(a, b)`	`a is b`
`assertIsNot(a, b)`	`a is not b`
`assertIsNone(x)`	`x is None`
`assertIsNotNone(x)`	`x is not None`
`assertIn(a, b)`	`a in b`
`assertNotIn(a, b)`	`a not in b`
`assertIsInstance(a, b)`	`isinstance(a, b)`
`assertNotIsInstance(a, b)`	`not isinstance(a, b)`

Assert Methods



Method	Checks that
`assertAlmostEqual(a, b)`	`round(a-b, 7) == 0`
`assertNotAlmostEqual(a, b)`	`round(a-b, 7) != 0`
`assertGreater(a, b)`	`a > b`
`assertGreaterEqual(a, b)`	`a >= b`
`assertLess(a, b)`	`a < b`
`assertLessEqual(a, b)`	`a <= b`
`assertRegex(s, r)`	`r.search(s)`
`assertNotRegex(s, r)`	`not r.search(s)`
`assertCountEqual(a, b)`	*a* and *b* have the same elements in the same number, regardless of their order.

Assert Methods



Method	Used to compare
`assertMultiLineEqual(a, b)`	strings
`assertSequenceEqual(a, b)`	sequences
`assertListEqual(a, b)`	lists
`assertTupleEqual(a, b)`	tuples
`assertSetEqual(a, b)`	sets or frozensets
`assertDictEqual(a, b)`	dicts

References

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