UnitTest Framework - Assertion

Python testing framework uses Python's built-in assert() function which tests a particular condition. If the assertion fails, an AssertionError will be raised. The testing framework will then identify the test as Failure. Other exceptions are treated as Error.

The following three sets of assertion functions are defined in unittest module −

- Basic Boolean Asserts
- Comparative Asserts
- Asserts for Collections

Basic assert functions evaluate whether the result of an operation is True or False. All the assert methods accept a **msg** argument that, if specified, is used as the error message on failure.

Sr.No.	Method & Description
1	**assertEqual(arg1, arg2, msg = None)** Test that *arg1* and *arg2* are equal. If the values do not compare equal, the test will fail.
2	**assertNotEqual(arg1, arg2, msg = None)** Test that *arg1* and *arg2* are not equal. If the values do compare equal, the test will fail.
3	**assertTrue(expr, msg = None)** Test that *expr* is true. If false, test fails
4	**assertFalse(expr, msg = None)** Test that *expr* is false. If true, test fails
5	**assertIs(arg1, arg2, msg = None)** Test that *arg1* and *arg2* evaluate to the same object.
6	**assertIsNot(arg1, arg2, msg = None)** Test that *arg1* and *arg2* don't evaluate to the same object.
7	**assertIsNone(expr, msg = None)** Test that *expr* is None. If not None, test fails
8	**assertIsNotNone(expr, msg = None)** Test that *expr* is not None. If None, test fails
9	**assertIn(arg1, arg2, msg = None)** Test that *arg1* is in *arg2*.
10	**assertNotIn(arg1, arg2, msg = None)** Test that *arg1* is not in *arg2*.

11	**assertIsInstance(obj, cls, msg = None)** Test that *obj* is an instance of *cls*
12	**assertNotIsInstance(obj, cls, msg = None)** Test that *obj* is not an instance of *cls*

Some of the above assertion functions are implemented in the following code −

Live Demo

```
import unittest

class SimpleTest(unittest.TestCase):
   def test1(self):
      self.assertEqual(4 + 5,9)
   def test2(self):
      self.assertNotEqual(5 * 2,10)
   def test3(self):
      self.assertTrue(4 + 5 == 9,"The result is False")
   def test4(self):
      self.assertTrue(4 + 5 == 10,"assertion fails")
   def test5(self):
      self.assertIn(3,[1,2,3])
   def test6(self):
      self.assertNotIn(3, range(5))

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

When the above script is run, test2, test4 and test6 will show failure and others run successfully.

```
FAIL: test2 (__main__.SimpleTest)
----------------------------------------------------------------------
Traceback (most recent call last):
   File "C:\Python27\SimpleTest.py", line 9, in test2
      self.assertNotEqual(5*2,10)
AssertionError: 10 == 10

FAIL: test4 (__main__.SimpleTest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Python27\SimpleTest.py", line 13, in test4
    self.assertTrue(4+5==10,"assertion fails")
AssertionError: assertion fails

FAIL: test6 (__main__.SimpleTest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Python27\SimpleTest.py", line 17, in test6
    self.assertNotIn(3, range(5))
AssertionError: 3 unexpectedly found in [0, 1, 2, 3, 4]

----------------------------------------------------------------------
Ran 6 tests in 0.001s

FAILED (failures = 3)
```

The second set of assertion functions are **comparative asserts −**

- **assertAlmostEqual** (first, second, places = 7, msg = None, delta = None)

  Test that *first* and *second* are approximately (or not approximately) equal by computing the difference, rounding to the given number of decimal *places* (default 7),

- **assertNotAlmostEqual** (first, second, places, msg, delta)

  Test that first and second are not approximately equal by computing the difference, rounding to the given number of decimal places (default 7), and comparing to zero.

  In both the above functions, if delta is supplied instead of places then the difference between first and second must be less or equal to (or greater than) delta.

  Supplying both delta and places raises a TypeError.

- **assertGreater** (first, second, msg = None)

  Test that *first* is greater than *second* depending on the method name. If not, the test will fail.

- **assertGreaterEqual** (first, second, msg = None)

  Test that *first* is greater than or equal to *second* depending on the method name. If not, the test will fail

- **assertLess** (first, second, msg = None)

  Test that *first* is less than *second* depending on the method name. If not, the test will fail

- **assertLessEqual** (first, second, msg = None)

Test that *first* is less than or equal to *second* depending upon the method name. If not, the test will fail.

- **assertRegexpMatches** (text, regexp, msg = None)

  Test that a regexp search matches the text. In case of failure, the error message will include the pattern and the text. regexp may be a regular expression object or a string containing a regular expression suitable for use by **re.search()**.

- **assertNotRegexpMatches** (text, regexp, msg = None)

  Verifies that a *regexp* search does not match *text*. Fails with an error message including the pattern and the part of *text* that matches. *regexp* may be a regular expression object or a string containing a regular expression suitable for use by re.search() .

The assertion functions are implemented in the following example −

Live Demo

```
import unittest
import math
import re

class SimpleTest(unittest.TestCase):
   def test1(self):
      self.assertAlmostEqual(22.0/7,3.14)
   def test2(self):
      self.assertNotAlmostEqual(10.0/3,3)
   def test3(self):
      self.assertGreater(math.pi,3)
   def test4(self):
      self.assertNotRegexpMatches("Tutorials Point (I) Private Limited","Point")

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

The above script reports test1 and test4 as Failure. In test1, the division of 22/7 is not within 7 decimal places of 3.14. Similarly, since the second argument matches with the text in first argument, test4 results in AssertionError.

```
=====================================================FAIL: test1 (__main__.SimpleTest)
----------------------------------------------------------------------
Traceback (most recent call last):
   File "asserttest.py", line 7, in test1
      self.assertAlmostEqual(22.0/7,3.14)
```

```
AssertionError: 3.142857142857143 != 3.14 within 7 places
================================================================
FAIL: test4 (__main__.SimpleTest)
----------------------------------------------------------------------
Traceback (most recent call last):
   File "asserttest.py", line 13, in test4
      self.assertNotRegexpMatches("Tutorials Point (I) Private Limited","Point")
AssertionError: Regexp matched: 'Point' matches 'Point' in 'Tutorials Point (I)
Private Limited'
----------------------------------------------------------------------

Ran 4 tests in 0.001s

FAILED (failures = 2)
```

Assert for Collections

This set of assert functions are meant to be used with collection data types in Python, such as List, Tuple, Dictionary and Set.

Sr.No.	Method & Description
1	**assertListEqual (list1, list2, msg = None)** Tests that two lists are equal. If not, an error message is constructed that shows only the differences between the two.
2	**assertTupleEqual (tuple1, tuple2, msg = None)** Tests that two tuples are equal. If not, an error message is constructed that shows only the differences between the two.
3	**assertSetEqual (set1, set2, msg = None)** Tests that two sets are equal. If not, an error message is constructed that lists the differences between the sets.
4	**assertDictEqual (expected, actual, msg = None)** Test that two dictionaries are equal. If not, an error message is constructed that shows the differences in the dictionaries.

The following example implements the above methods −

Live Demo

```
import unittest

class SimpleTest(unittest.TestCase):
   def test1(self):
      self.assertListEqual([2,3,4], [1,2,3,4,5])
   def test2(self):
      self.assertTupleEqual((1*2,2*2,3*2), (2,4,6))
   def test3(self):
      self.assertDictEqual({1:11,2:22},{3:33,2:22,1:11})

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

In the above example, test1 and test3 show AssertionError. Error message displays the differences in List and Dictionary objects.

```
FAIL: test1 (__main__.SimpleTest)
----------------------------------------------------------------------
Traceback (most recent call last):
   File "asserttest.py", line 5, in test1
      self.assertListEqual([2,3,4], [1,2,3,4,5])
AssertionError: Lists differ: [2, 3, 4] != [1, 2, 3, 4, 5]

First differing element 0:
2
1

Second list contains 2 additional elements.
First extra element 3:
4

- [2, 3, 4]
+ [1, 2, 3, 4, 5]
? +++       +++

FAIL: test3 (__main__.SimpleTest)
----------------------------------------------------------------------
Traceback (most recent call last):
   File "asserttest.py", line 9, in test3
      self.assertDictEqual({1:11,2:22},{3:33,2:22,1:11})
AssertionError: {1: 11, 2: 22} != {1: 11, 2: 22, 3: 33}
```

```
- {1: 11, 2: 22}
+ {1: 11, 2: 22, 3: 33}
?               +++++++

----------------------------------------------------------------------
Ran 3 tests in 0.001s

FAILED (failures = 2)
```