UnitTest Framework - Skip Test

Support for skipping tests has been added since Python 2.7. It is possible to skip individual test method or TestCase class, conditionally as well as unconditionally. The framework allows a certain test to be marked as an 'expected failure'. This test will 'fail' but will not be counted as failed in TestResult.

To skip a method unconditionally, the following unittest.skip() class method can be used −

```
import unittest

   def add(x,y):
      return x+y

class SimpleTest(unittest.TestCase):
   @unittest.skip("demonstrating skipping")
   def testadd1(self):
      self.assertEquals(add(4,5),9)

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

Since skip() is a class method, it is prefixed by @ token. The method takes one argument: a log message describing the reason for the skip.

When the above script is executed, the following result is displayed on console −

```
C:\Python27>python skiptest.py
s
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK (skipped = 1)
```

The character 's' indicates that a test has been skipped.

Alternate syntax for skipping test is using instance method skipTest() inside the test function.

```
def testadd2(self):
   self.skipTest("another method for skipping")
```

```
self.assertTrue(add(4 + 5) == 10)
```

The following decorators implement test skipping and expected failures −

S.No.	Method & Description
1	**unittest.skip(reason)** Unconditionally skip the decorated test. *reason* should describe why the test is being skipped.
2	**unittest.skipIf(condition, reason)** Skip the decorated test if condition is true.
3	**unittest.skipUnless(condition, reason)** Skip the decorated test unless condition is true.
4	**unittest.expectedFailure()** Mark the test as an expected failure. If the test fails when run, the test is not counted as a failure.

The following example demonstrates the use of conditional skipping and expected failure.

Live Demo

```
import unittest

class suiteTest(unittest.TestCase):
   a = 50
   b = 40

   def testadd(self):
      """Add"""
      result = self.a+self.b
      self.assertEqual(result,100)

   @unittest.skipIf(a>b, "Skip over this routine")
   def testsub(self):
      """sub"""
```

```
        result = self.a-self.b
        self.assertTrue(result == -10)

    @unittest.skipUnless(b == 0, "Skip over this routine")
    def testdiv(self):
        """div"""
        result = self.a/self.b
        self.assertTrue(result == 1)

    @unittest.expectedFailure
    def testmul(self):
        """mul"""
        result = self.a*self.b
        self.assertEqual(result == 0)

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

In the above example, testsub() and testdiv() will be skipped. In the first case a>b is true, while in the second case b == 0 is not true. On the other hand, testmul() has been marked as expected failure.

When the above script is run, two skipped tests show 's' and the expected failure is shown as 'x'.

```
C:\Python27>python skiptest.py
Fsxs
======================================================================
FAIL: testadd (__main__.suiteTest)
Add
----------------------------------------------------------------------
Traceback (most recent call last):
   File "skiptest.py", line 9, in testadd
      self.assertEqual(result,100)
AssertionError: 90 != 100

----------------------------------------------------------------------
Ran 4 tests in 0.000s

FAILED (failures = 1, skipped = 2, expected failures = 1)
```