How to Skip Tests

The unittest module supports skipping tests as of Python 3.1. There are a few use cases for skipping tests:

> - You might want to skip a test if the version of a library doesn't support what you want to test
> - The test is dependent on the operating system it is running under
> - Or you have some other criteria for skipping a test

Let's change our test case so it has a couple of tests that will be skipped:

```
import mymath
import sys
import unittest

class TestAdd(unittest.TestCase):
    """
    Test the add function from the mymath module
    """

    def test_add_integers(self):
        """
        Test that the addition of two integers returns the correct total
        """
        result = mymath.add(1, 2)
        self.assertEqual(result, 3)

    def test_add_floats(self):
        """
        Test that the addition of two floats returns the correct result
        """
        result = mymath.add(10.5, 2)
        self.assertEqual(result, 12.5)

    @unittest.skip('Skip this test')
    def test_add_strings(self):
        """
        Test the addition of two strings returns the two string as one
        concatenated string
        """
        result = mymath.add('abc', 'def')
        self.assertEqual(result, 'abcdef')
    @unittest.skipUnless(sys.platform.startswith("win"), "requires Windows")
    def test_adding_on_windows(self):

        result = mymath.add(1, 2)
        self.assertEqual(result, 3)
```

Here we demonstrate two different methods of skipping a test: **skip** and **skipUnless**. You will notice that we are decorating the functions that need to be skipped. The **skip** decorator can be used to skip any test for any reason. The **skipUnless** decorator will skip a test unless the condition returns True. So if you run this test on Mac or Linux, it will get skipped. There is also a **skipIf** decorator that will skip a test if the condition is True.

You can run this script with the verbose flag to see why it's skipping tests:

```
python -m unittest test_mymath.py -v
```

This command will result in the following output:

```
test_add_floats (test_mymath4.TestAdd) ... ok
test_add_integers (test_mymath4.TestAdd) ... ok
test_add_strings (test_mymath4.TestAdd) ... skipped 'Skip this test'
test_adding_on_windows (test_mymath4.TestAdd) ... skipped 'requires Windows'

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

OK (skipped=2)
```

This output tells us that we attempted to run four tests, but skipped two of them.

There is also an **expectedFailure** decorator that you can add to a test that you know will fail. I'll leave that one for you to try out on your own.