Coverage.py

Coverage.py is a 3rd party tool for Python that is used for measuring your code coverage. It was originally created by Ned Batchelder. The term "coverage" in programming circles is typically used to describe the effectiveness of your tests and how much of your code is actually covered by tests. You can use coverage.py with Python 2.6 up to the current version of Python 3 as well as with PyPy.

If you don't have coverage.py installed, you may do so using pip:

```
pip install coverage
```

Now that we have coverage.py installed, we need some code to use it with. Let's use the **mymath** module that we created earlier in the book. Here's the code:

```
def add(a, b):
    return a + b


def subtract(a, b):
    return a - b


def multiply(a, b):
    return a * b


def divide(numerator, denominator):
    return float(numerator) / denominator
```

Now we need a test. Fortunately we wrote a simple test in the unit testing chapter. To refresh your memory, here is the code for test:

```
# test_mymath.py
import mymath
import unittest
```

```
import unittest

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

    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)

    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')


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

Now that we have all the pieces, we can run coverage.py using the test. Open up a terminal and navigate to the folder that contains the **mymath** module and the test code we wrote. Now we can call coverage.py like this:

```
coverage run test_mymath.py
```

Note that we need to call **run** to get coverage.py to run the module. If your module accepts arguments, you can pass those in as you normally would. When you do this, you will see the test's output as if you ran it yourself. You will also find a new file in the directory that is called **.coverage**. To get information out of this file, you will need to run the following command:

```
coverage report -m
```

Executing this command will result in the following output:

```
Name            Stmts   Miss  Cover   Missing
```

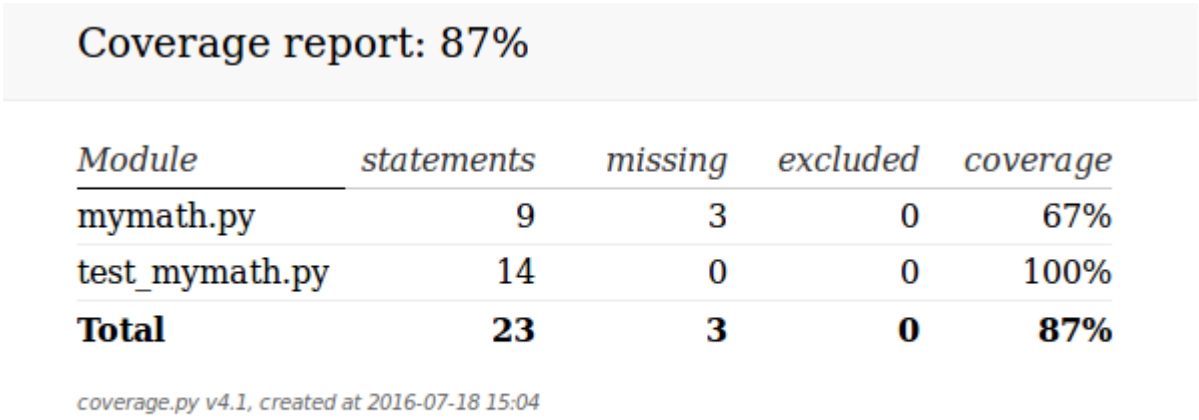
```
----------------------------------------
mymath.py             9      3    67%   9, 13, 17
test_mymath.py       14      0   100%

----------------------------------------
TOTAL                23      3    87%
```

The **-m** flag tells coverage.py that you want it to include the **Missing** column in the output. If you omit the **-m**, then you'll only get the first four columns. What you see here is that coverage ran the test code and determined that I have only 67% of the mymath module covered by my unit test. The "Missing" column tells me what lines of code still need coverage. If you look at the lines coverage.py points out, you will quickly see that my test code doesn't test the **subtract**, **multiply** or **divide** functions.

Before we try to add more test coverage, let's learn how to make coverage.py produce an HTML report. To do this, all you need to do is run the following command:

```
coverage html
```

This command will create a folder named **htmlcov** that contains various files. Navigate into that folder and try opening **index.html** in your browser of choice. On my machine, it loaded a page like this:

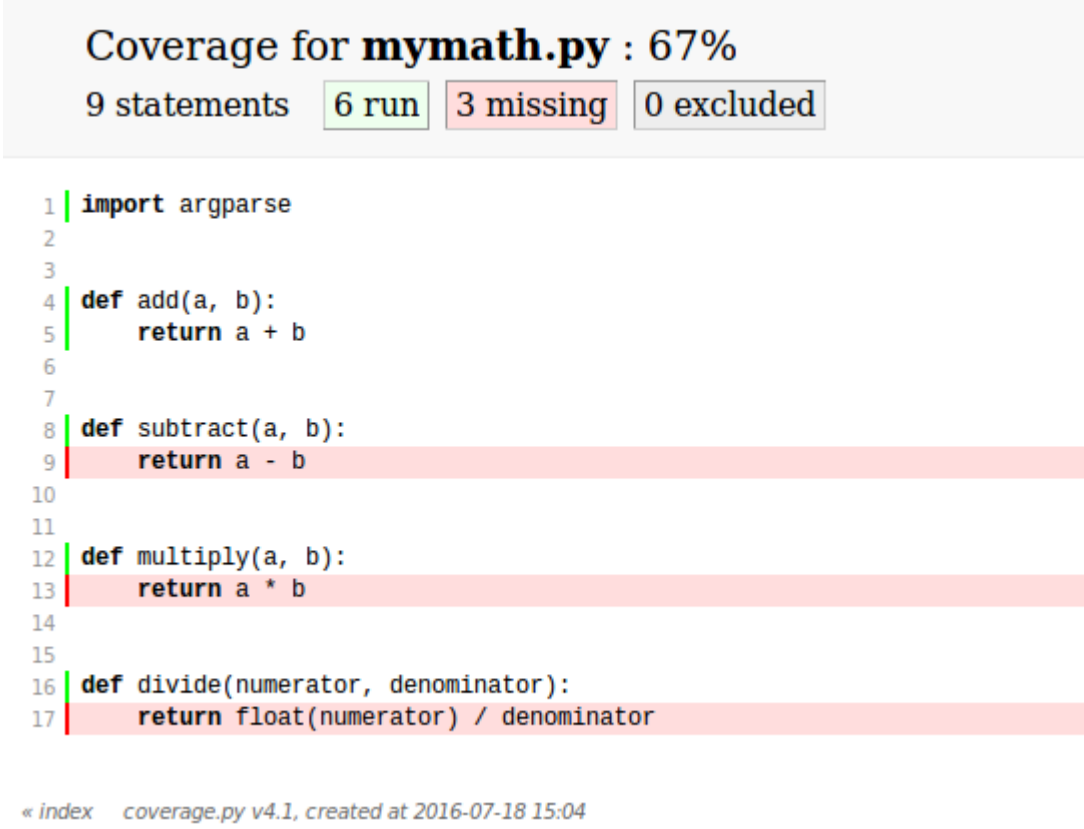


Coverage report: 87%

Module	*statements*	*missing*	*excluded*	*coverage*
mymath.py	9	3	0	67%
test_mymath.py	14	0	0	100%
Total	**23**	**3**	**0**	**87%**

coverage.py v4.1, created at 2016-07-18 15:04

You can actually click on the modules listed to load up an annotated web page that shows you what parts of the code are not covered. Since the **mymath.py** module obviously isn't covered very well, let's click on that one. You should end up seeing something like the following:



This screenshot clearly shows what parts of the code were not covered in our original unit test. Now that we know definitively what's missing in our test coverage, let's add a unit test for our **subtract** function and see how that changes things!

Open up your copy of **test_mymath.py** and add the following class to it:

```
class TestSubtract(unittest.TestCase):
    """
    Test the subtract function from the mymath library
    """


    def test_subtract_integers(self):
        """
        Test that subtracting integers returns the correct result
        """
        result = mymath.subtract(10, 8)
        self.assertEqual(result, 2)
```

Now we need to re-run coverage against the updated test. All you need to do is re-run this command: **coverage run test_mymath.py**. The output will show that four tests have passed successfully. Now re-run **coverage html** and re-open the "index.html" file. You should now see the that we're at 78% coverage:

Coverage report: 93%

Module	statements	missing	excluded	coverage
mymath.py	9	2	0	78%
test_mymath.py	18	0	0	100%
Total	**27**	**2**	**0**	**93%**

coverage.py v4.1, created at 2016-07-18 16:08

This is an 11% improvement! Let's go ahead and add a simple test for the multiply and divide functions and see if we can hit 100% coverage!

```
class TestMultiply(unittest.TestCase):
    """
    Test the multiply function from the mymath library
    """

    def test_subtract_integers(self):
        """
        Test that multiplying integers returns the correct result
        """
        result = mymath.multiply(5, 50)
        self.assertEqual(result, 250)


class TestDivide(unittest.TestCase):
    """
    Test the divide function from the mymath library
    """

    def test_divide_by_zero(self):
        """
        Test that multiplying integers returns the correct result
        """
        with self.assertRaises(ZeroDivisionError):
            result = mymath.divide(8, 0)
```

Now you can re-run the same commands as before and reload the "index.html" file. When you do, you should see something like the following:

Coverage report: 100%

Module	*statements*	*missing*	*excluded*	*coverage*
mymath.py	9	0	0	100%
test_mymath.py	26	0	0	100%
Total	**35**	**0**	**0**	**100%**

coverage.py v4.1, created at 2016-07-18 16:20

As you can see, we have hit full test coverage! Of course, full coverage in this case means that each function is exercised by our test suite. The problem with this is that we have three times the number of tests for the addition function versus the others, but coverage.py doesn't give us any kind of data about that. However it will give us a good idea of basic coverage even if it can't tell us if we've tested every possible argument permutation imaginable.