# A Simple Example

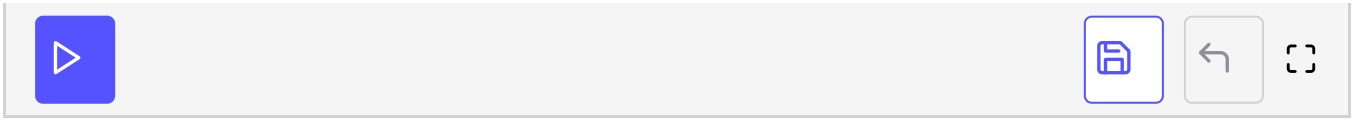
I always find a code example or two to be the quickest way to learn how something new works. So let's create a little module that we will call **mymath.py**. Then put the following code into it:

```
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
```

This module defines four mathematical functions: add, subtract, multiply and divide. They do not do any error checking and they actually don't do exactly what you might expect. For example, if you were to call the **add** function with two strings, it would happily concatenate them together and return them. But for the purposes of illustration, this module will do for creating a test case. So let's actually write a test case for the add function! We will call this script **test_mymath.py** and save it in the same folder that contains **mymath.py**.

```
import mymath
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()
```



Let's take a moment and go over how this code works. First we import our mymath module and Python's **unittest** module. Then we subclass **TestCase** and add three tests, which translates into three methods. The first function tests the addition of two integers; the second function tests the addition of two floating point numbers; and the last function concatenates two strings together. Finally we call unittest's **main** method at the end.

You will note that each method begins with the letters "test". This is actually important! It tells the test runner which methods are tests that it should run. Each test should have at least one assert which will verify that the result is as we expected. The unittest module supports many different types of asserts. You can test for exceptions, for Boolean conditions, and for many other conditions.

Let's try running out test. Open up a terminal and navigate to the folder that contains your mymath module and your test module:

```
python test_mymath.py
```

This will execute our test and we should get the following output:

```
...
----------------------------------------------------------------------
Ran 3 tests in 0.001s


OK
```

You will note that there are three periods. Each period represents a test that has passed. Then it tells us that it ran 3 tests, the time it took and the result: OK. That tells us that all the tests passed successfully.

You can make the output a bit more verbose by passing in the **-v** flag:

```
python test_mymath.py -v
```

This will cause the following output to be printed to stdout:

```
test_add_floats (__main__.TestAdd) ... ok
test_add_integers (__main__.TestAdd) ... ok
test_add_strings (__main__.TestAdd) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

As you can see, this shows us exactly what tests were run and the results of each test. This also leads us into our next section where we will learn about some of the commands we can use with unittest on the command line.