A quick intro to Unittest in Python

Aims

- To be introduced to the Unittest module.
- To learn how to create and run tests.

Requirements

- A computer.
- A working install of Python.

Resources

https://docs.python.org/3/library/unittest.html

https://www.freecodecamp.org/news/unit-testing-in-python/

https://www.geeksforgeeks.org/unit-testing-python-unittest/

https://realpython.com/python-unittest/

Why Do We Test

With any code you create you have to trust that it works and that it will do exactly what you want it to.

However in the real world this does not always work out how we intend and it's normally wise to test the code be before deployment.

We can write tests that isolate sections of our code and verify its functioning as intended. Also a well-written battery or suite of tests can also serve as documentation for the project at hand.

Python has a built in test suite called *unittest* and we will use this to develop our understanding of testing.

Unittest uses the concept of *assertions* to assert that a given function / code snippet will result in a given result.

Below are some of the methods that are commonly used to write assertions:

Method	Description
.assertEqual(a, b)	Checks if a is equal to b, similar to the expression a == b.
.assertTrue(x)	Asserts that the boolean value of x is True, equivalent to bool(x) is True.
.assertIsInstance(a, b) Asserts that a is an instance of class b, similar to the expression isinstance(a, b).	.assertIsNone(x)
Ensures that x is None, similar to the expression x is None.	.assertFalse(x)
Asserts that the boolean value of x is False, similar to bool(x) is False.	.assertIs(a, b)
Verifies if a is identical to b, akin to the expression a is b.	.assertIn(a, b)

Our First Test

Using the example from the freecodecamp link we first need to create a simple calculator file:

1. Create a directory called unit-tests
2. Using IDLE create a python file in unit-tests named *calculator.py*
3. Enter the following code into the file:

```
def add(x, y):
    """add numbers"""
    return x + y

def subtract(x, y):
    """subtract numbers"""
    return x - y

def divide(x, y):
    """divide numbers"""
    return x / y

def multiply(x, y):
    """multiply numbers"""
    return x * y
```

4. Then create a file called *test_calculator.py* and enter the following code:

```
import unittest
import calculator

class TestCalculator(unittest.TestCase):
    def test_add(self):
        self.assertEqual(calculator.add(1, 2), 3)
        self.assertEqual(calculator.add(-1, 1), 0)
        self.assertEqual(calculator.add(-1, -1), -2)
        self.assertEqual(calculator.add(0, 0), 0)
```

5. Run the code:

```
python -m unittest test_calculator.py
```

6. The code should output the following:

```
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

   This shows that all our tests have passed and code is working as confirmed.

7. Make the following change to *calculator.py* to make the *add* function subtract instead:

```
def add(x, y):
    """add numbers"""
    return x - y
```

8. Now when we run the tests as we did in step 5 we get the following output:

```
Traceback (most recent call last):
  File ".../test_calculator.py", line 6, in test_add
    self.assertEqual(calculator.add(1, 2), 3)
AssertionError: -1 != 3

----------------------------------------------------------------------
Ran 1 test in 0.000s
```

   Which shows that our code has failed testing and that we need to investigate.

Making Our Test Script A Standalone Executable

You may have noticed that we need to specify the *unittest* module when we execute our test script.

```
python -m unittest test_calculator.py
```

There is nothing really wrong with this as it works as intended. However it would be better if we could run it directley like so:

```
python test_calculator.py
```

To do so we need to add a two lines tot the bottom of *test_calculator.py*:

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

Once save and if we now execute:

```
python test_calculator.py
```

Then our testfile will execute as expected.

The basics of these lines is that the *if* statement detects that the script is being ran standalone then it will, in this case, execute *unittest.main()* causing the test class to be executed.

If it’s detected that it’s being imported then the Class, methods and functions will be made availible to the script that is importing.

As an experiment try running the file directly with and without the lines.

This can be used in anyfile.

Further reading on this can be found here:

https://docs.python.org/3/library/__main__.html