Unit Testing with Python

PowerPoint Presentation in Software Engineering and Testing

What is Test automation?

- Notice that this *isn't just testing*, *its Automated testing*.
- We know the importance of testing, but just one person testing your code is inefficient, and will make errors.
- So we need someone to do it for us. Consistently, and reliably every single time we need to publish software.
- Automated tests exist for this reason. That is what we will be looking at in upcoming slides.

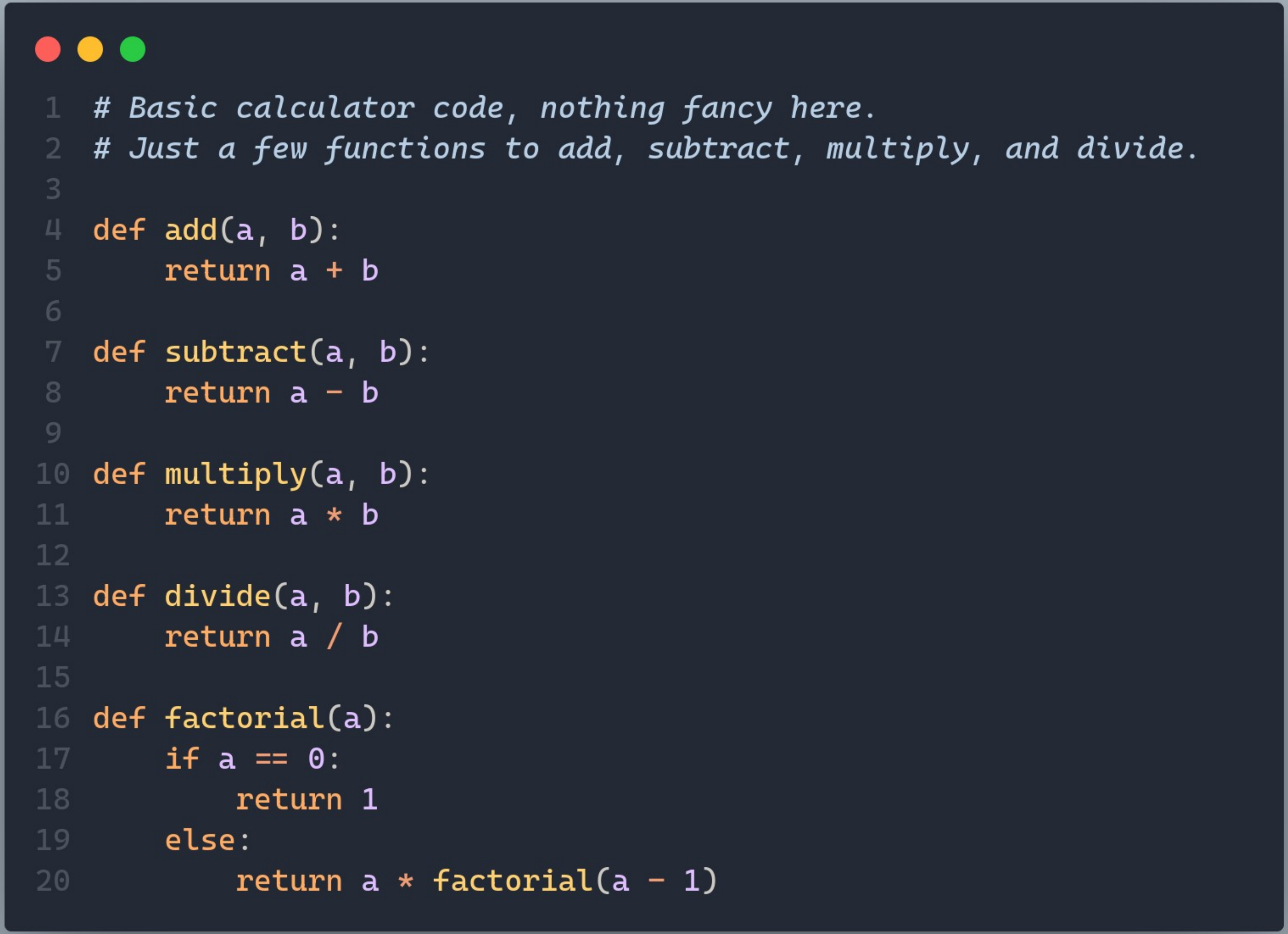
What is the unittest library in python?

- It is a built in library in python, that provides some functions and classes so that it becomes easy to write automated tests for your code.

Are there other libraries?

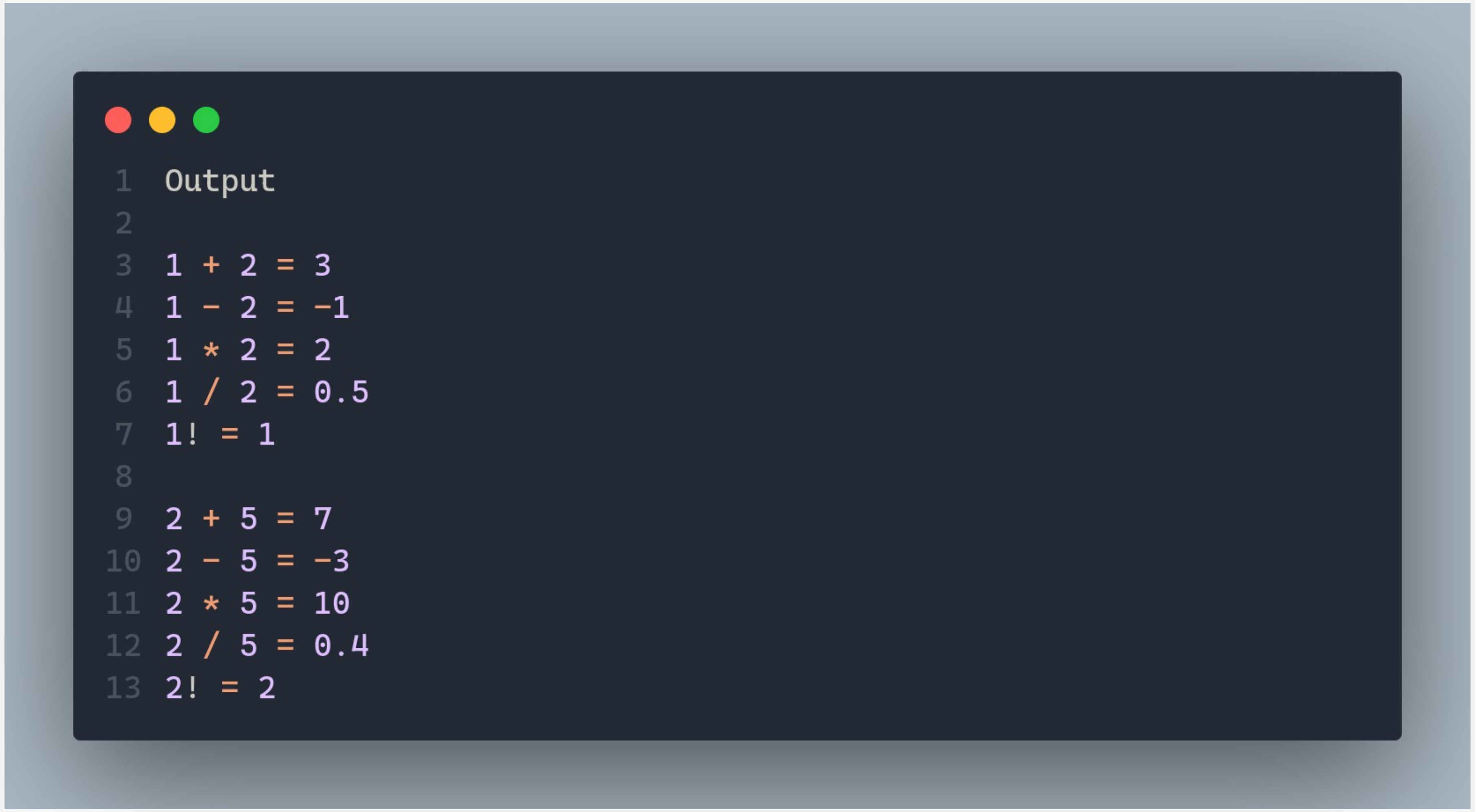
- Yes, there are many other libraries in python. Pytest is one of them. You also may wanna consider Java and C# for writing these automated tests for your code, as they also have testing frameworks written for them.
- Java provides a framework called **JUnit** to perform the unit testing of our Java code. In the development of **test-driven** development, JUnit is very important. The JUnit is one of the frameworks available in the unit testing frameworks. The **xUnit** is the unit testing framework family, and JUnit is the part of the **xUnit**.
- of the code. It also increases the productivity of the programmer. JUnit promotes the idea of "first testing then coding", which emphasizes setting test data for a piece of code that can be tested first and then implemented. Junit increases the stability

Consider this example



```python
# Basic calculator code, nothing fancy here.
# Just a few functions to add, subtract, multiply, and divide.

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

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

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

def divide(a, b):
    return a / b

def factorial(a):
    if a == 0:
        return 1
    else:
        return a * factorial(a - 1)
```

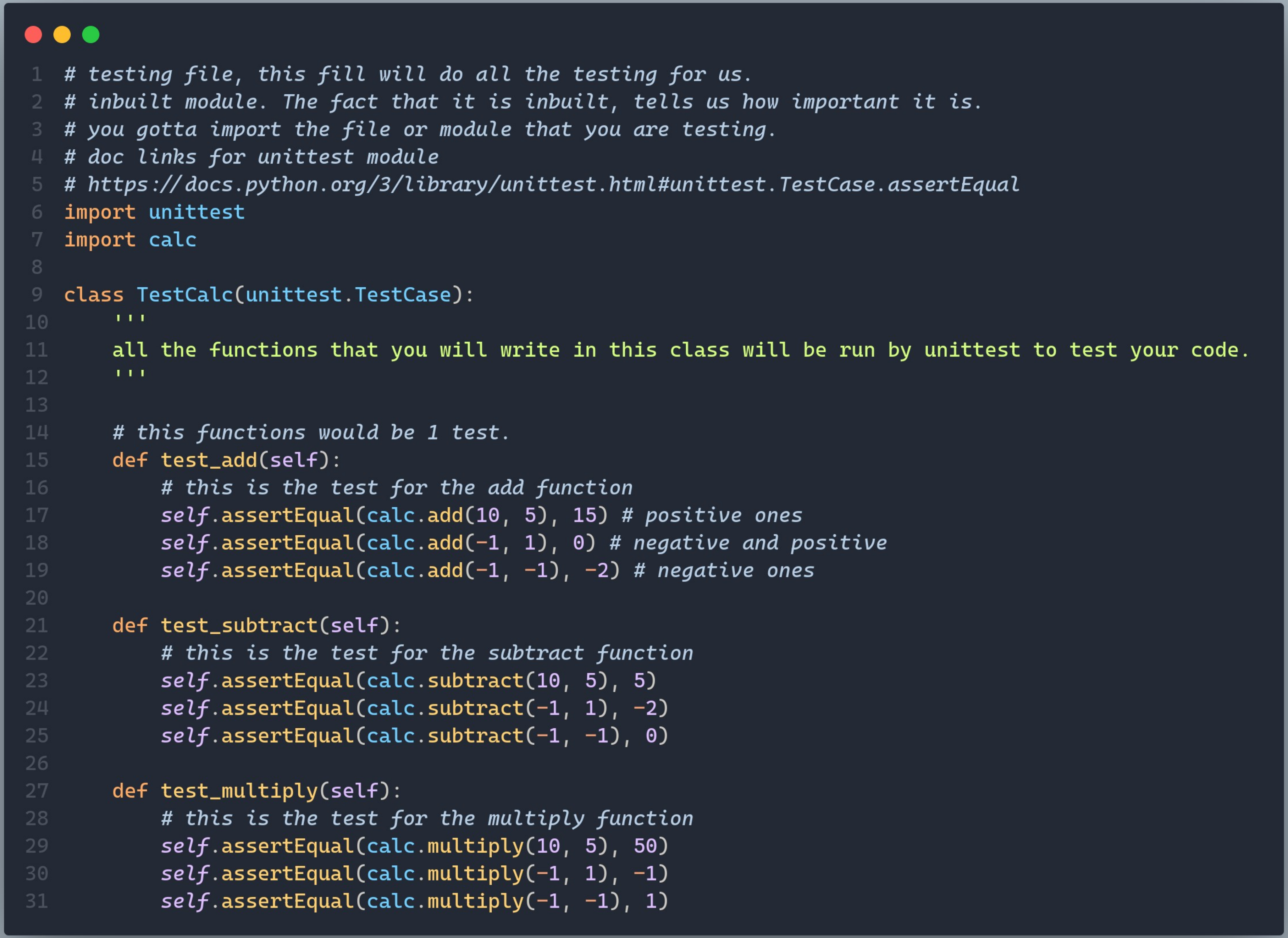
If you run that file with some example, this would be the output.



```
Output

1 + 2 = 3
1 - 2 = -1
1 * 2 = 2
1 / 2 = 0.5
1! = 1

2 + 5 = 7
2 - 5 = -3
2 * 5 = 10
2 / 5 = 0.4
2! = 2
```

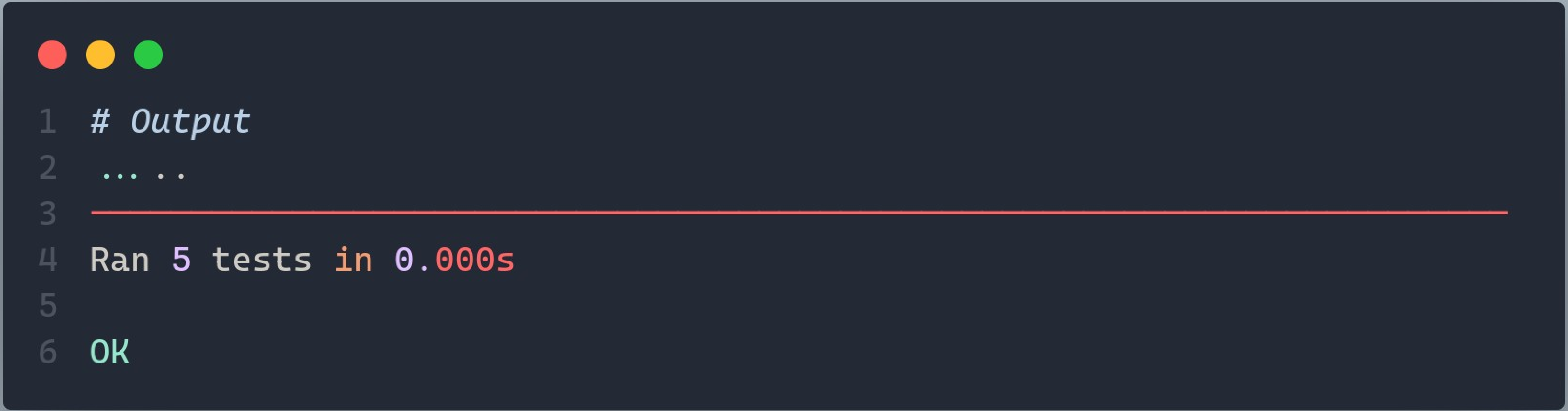
On the Next slide we have the testing file, this code would be saved as a separate file, that would be next to your main code.

You can then import your code into your testing file, and then write as many tests as your code needs, but in a separate file to maintain some modularity in your code.



```python
# testing file, this fill will do all the testing for us.
# inbuilt module. The fact that it is inbuilt, tells us how important it is.
# you gotta import the file or module that you are testing.
# doc links for unittest module
# https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertEqual
import unittest
import calc

class TestCalc(unittest.TestCase):
    '''
    all the functions that you will write in this class will be run by unittest to test your code.
    '''

    # this functions would be 1 test.
    def test_add(self):
        # this is the test for the add function
        self.assertEqual(calc.add(10, 5), 15) # positive ones
        self.assertEqual(calc.add(-1, 1), 0) # negative and positive
        self.assertEqual(calc.add(-1, -1), -2) # negative ones

    def test_subtract(self):
        # this is the test for the subtract function
        self.assertEqual(calc.subtract(10, 5), 5)
        self.assertEqual(calc.subtract(-1, 1), -2)
        self.assertEqual(calc.subtract(-1, -1), 0)

    def test_multiply(self):
        # this is the test for the multiply function
        self.assertEqual(calc.multiply(10, 5), 50)
        self.assertEqual(calc.multiply(-1, 1), -1)
        self.assertEqual(calc.multiply(-1, -1), 1)
```

And this would be the output for the test.



```
# Output
... ..
______________________________________________________________________
Ran 5 tests in 0.000s

OK
```

So what is the point of all this?

- Imagine a situation where you are building something like google chrome. You want it to work on all platforms, and you cant afford to mess up the code.
- In such a scenario, after you write some code, you wont just publish it as the next release and prompt billions of users world wide for an update, without making sure it works, what if you made a logical error while coding?

- To solve this universal issue in software engineering, you would naturally want to test your code, now again it is unreasonable to just run it again and again after every change you make. Which is why such libraries exist.
- They offer reliability as well, if someone knows you have tested your code with unittest, or pytest, then they can trust your codebase, and you have produced ‘good’ software.

Will you do this as part of your job?

Most of the test creation will be done by AI in the very near future, but it will be indisputably prone to errors despite of its vast knowledge base.

There will always be a real world scenario that AI will not be able to account for, but You as the developer will have to.

For this it is essential to understand why and how EXACTLY software testing is done.