Hands-on Lab: Unit Testing

Unit Testing Lab

Estimated time needed: 30 minutes

Objectives

After completing this lab you will be able to:

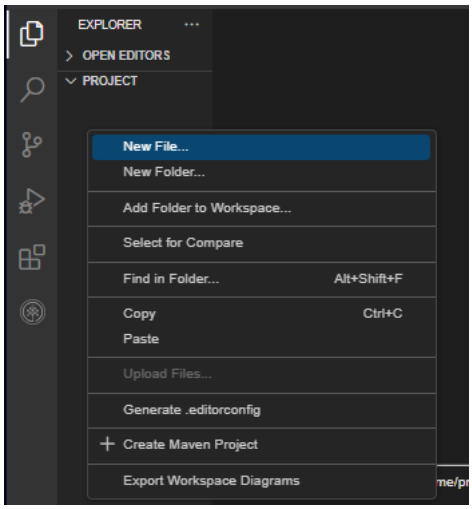
- Write unit tests to test a function.
- Run unit tests and interpret the results.

About the lab environment

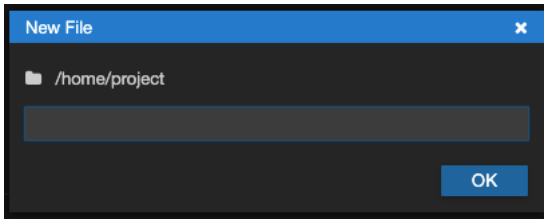
Cloud IDE is an open-source IDE(Integrated Development Environment), that can be run on desktop or on cloud. You will be using the Cloud IDE to do this lab. When you log into the Cloud IDE environment, you are presented with a 'dedicated computer on the cloud' exclusively for you. This is available to you as long as you work on the labs. Once you log off, this 'dedicated computer on the cloud' is deleted along with any files you may have created. So, it is a good idea to finish your labs in a single session. If you finish part of the lab and return to the Theia lab later, you may have to start from the beginning. Plan to work out all your Theia labs when you have the time to finish the complete lab in a single session.

Create a new python file named mymodule.py

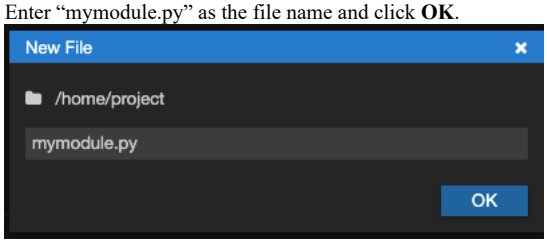
On the window, Right click on the **Explorer** and select **New File** option, as shown in the image below.



A pop up appears with title **New File**, as shown in the image below.



Enter “mymodule.py” as the file name and click **OK**.



A file “mymodule.py” will be created for you.

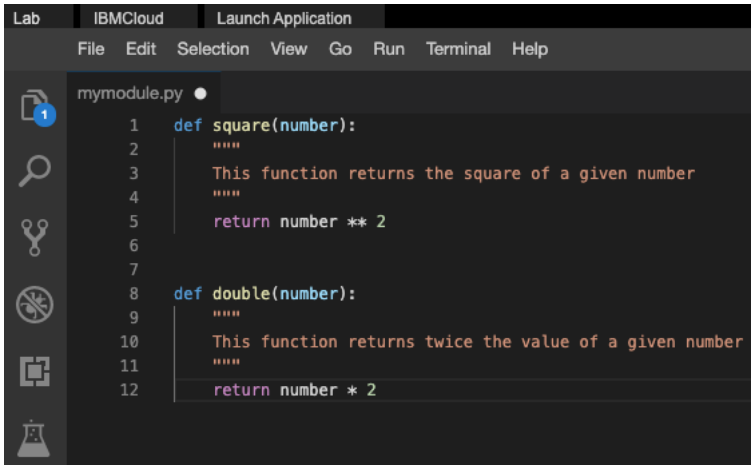
You are now ready to add code to mymodule.py

Copy and paste the below code into mymodule.py

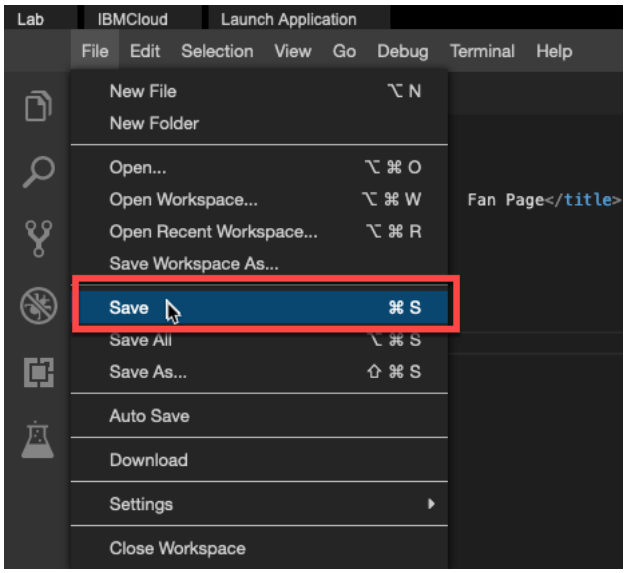
```
def square(number):
    """
```

```
    This function returns the square of a given number
    """
    return number ** 2
def double(number):
    """
    This function returns twice the value of a given number
    """
    return number * 2
```

You should see a screen like this now.



Save the file by using the Save option in the File Menu.



Write Unit Tests

Write the unit tests for square function

Let us write test cases for these three scenarios.

- When 2 is given as input the output must be 4.
- When 3.0 is given as input the output must be 9.0.
- When -3 is given as input the output must not be -9.

Write the unit tests for double function

Let us write test cases for these three scenarios.

- When 2 is given as input the output must be 4.
- When -3.1 is given as input the output must be -6.2.
- When 0 is given as input the output must be 0.

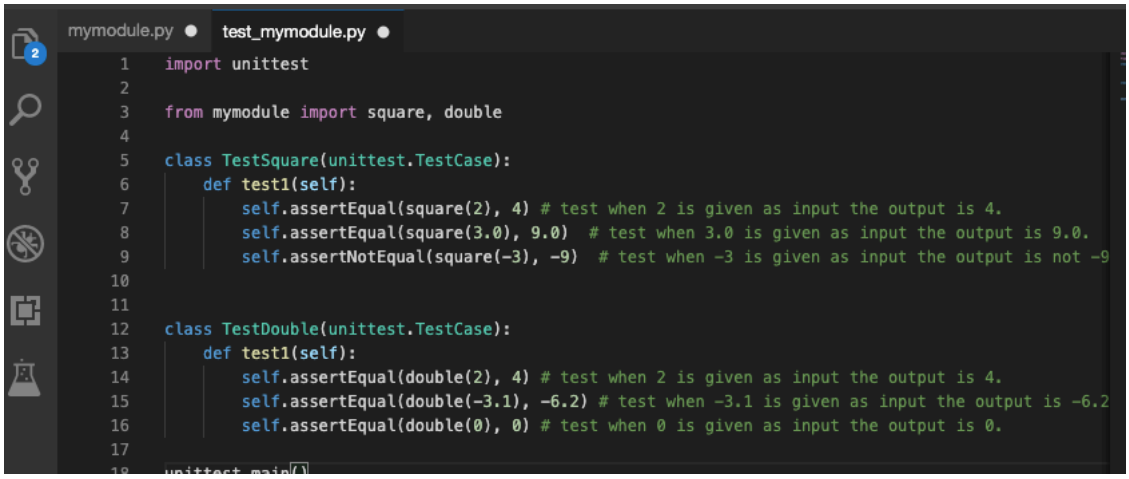
Create a new file and name it as test_mymodule.py

Copy and paste the below code into test_mymodule.py

```
# Import the 'unittest' module to create unit tests for your code.
import unittest
# Import the 'square' and 'double' functions from the 'mymodule' module.
from mymodule import square, double
# Define a test case class for testing the 'square' function.
# A test case is a single unit of testing. It checks a specific aspect of the code's behavior.
class TestSquare(unittest.TestCase):
```

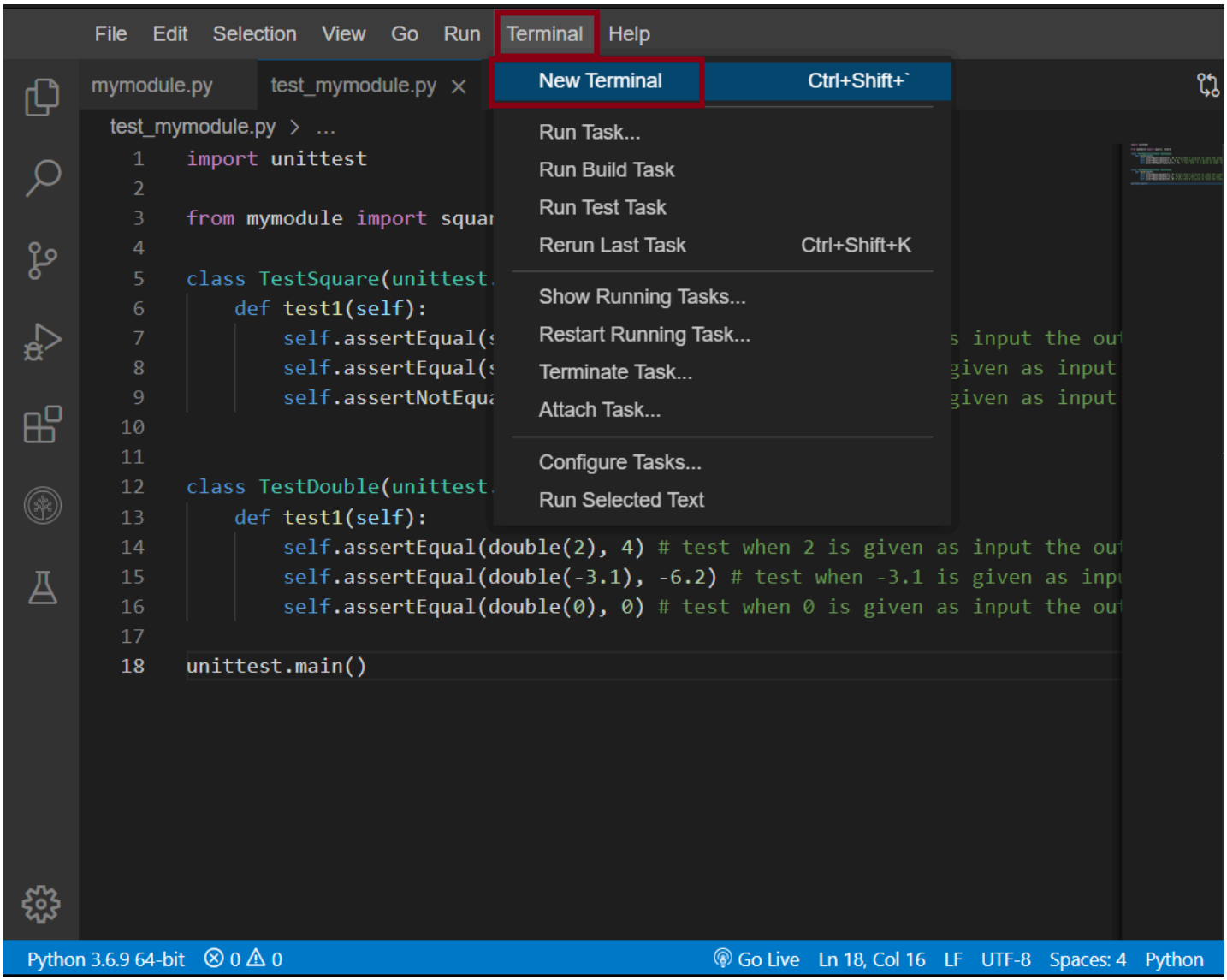
```
        # Define the first test method for the 'square' function.
        # Test methods should start with the word 'test' so that the test runner recognizes them as test cases.
        def test1(self):
            # Check that calling 'square(2)' returns 4.
            # This tests if the function correctly computes the square of 2.
            self.assertEqual(square(2), 4) # test when 2 is given as input the output is 4.
            # Check that calling 'square(3.0)' returns 9.0.
            # This tests if the function correctly computes the square of 3.0, verifying that it handles float inputs.
            self.assertEqual(square(3.0), 9.0)  # test when 3.0 is given as input the output is 9.0.
            # Check that calling 'square(-3)' does not return -9.
            # This tests that the function's output is not -9, verifying that the square of -3 should be 9.
            self.assertNotEqual(square(-3), -9)  # test when -3 is given as input the output is not -9.
    # Define a test case class for testing the 'double' function.
    class TestDouble(unittest.TestCase):
        # Define the first test method for the 'double' function.
        def test1(self):
            # Check that calling 'double(2)' returns 4.
            # This tests if the function correctly computes double of 2.
            self.assertEqual(double(2), 4) # test when 2 is given as input the output is 4.
            # Check that calling 'double(-3.1)' returns -6.2.
            # This tests if the function correctly computes double of -3.1, verifying that it handles negative float inputs.
            self.assertEqual(double(-3.1), -6.2) # test when -3.1 is given as input the output is -6.2.
            # Check that calling 'double(0)' returns 0.
            # This tests if the function correctly computes double of 0, verifying that the function works for edge cases.
            self.assertEqual(double(0), 0) # test when 0 is given as input the output is 0.

    # Run all the test cases defined in the module when the script is executed.
    # This will automatically discover and run all the test cases defined in the module.
    unittest.main()
```

You should see a screen like this now.

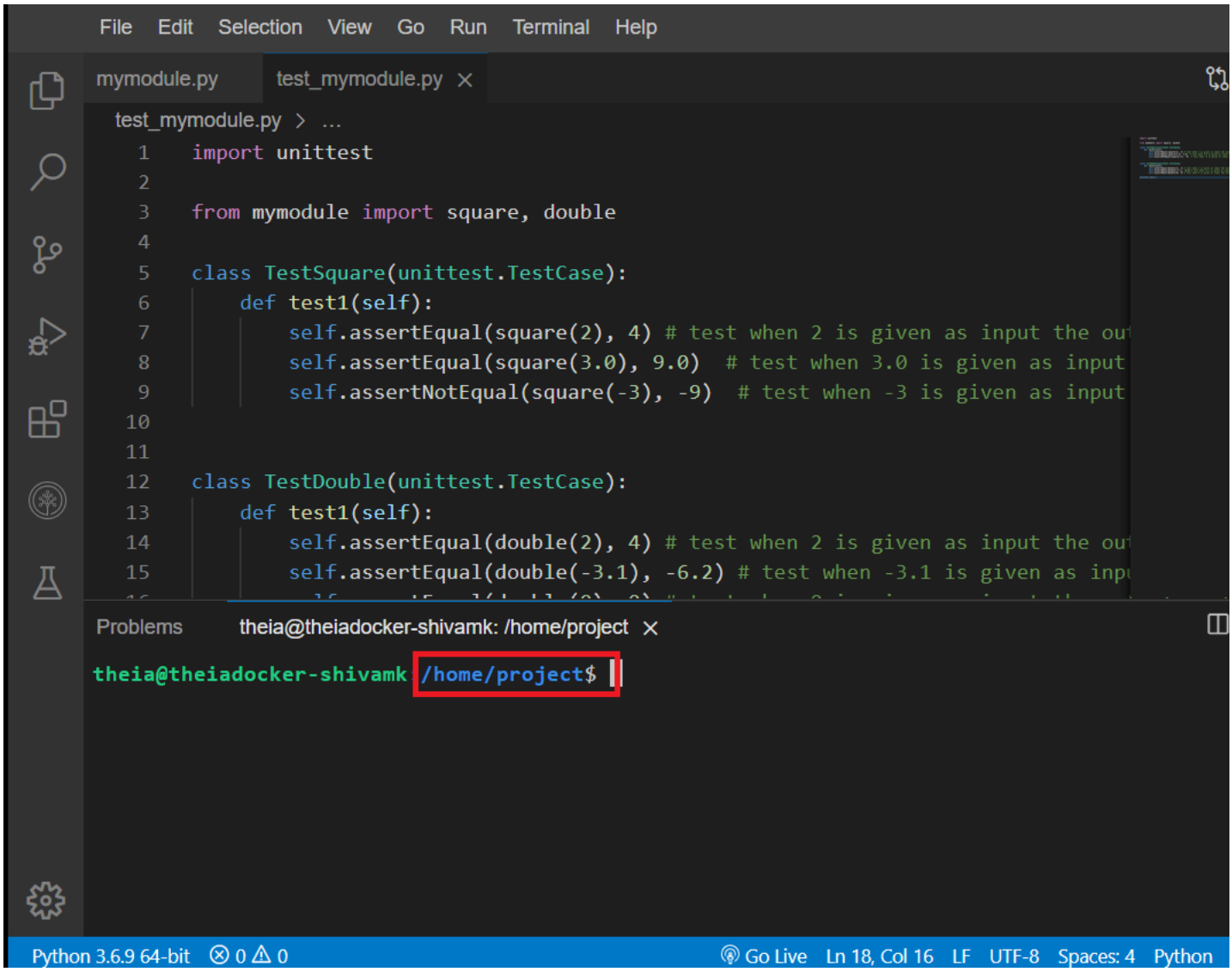


Run tests

To run tests, click on the “Terminal” and then click on the “New Terminal”

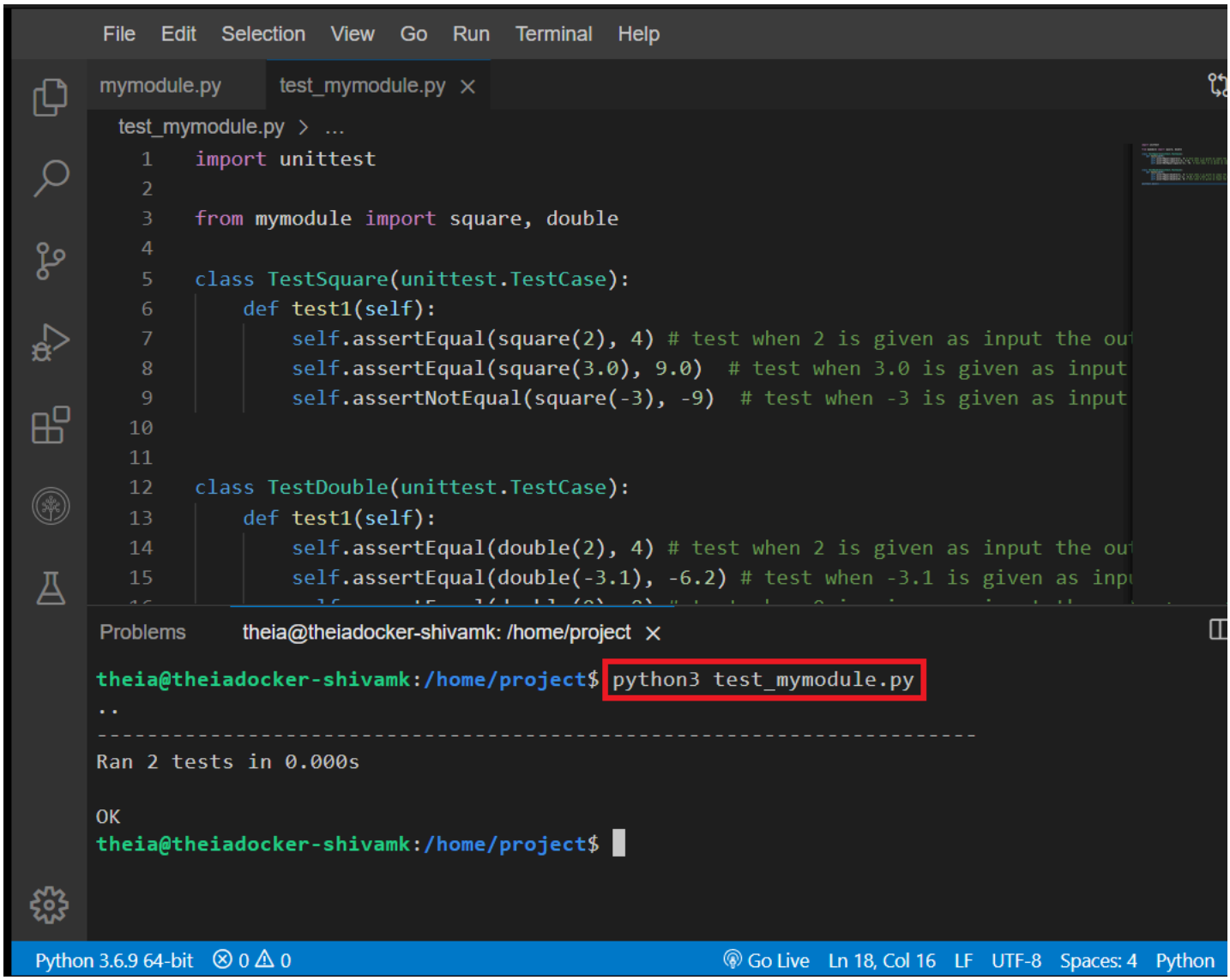


It will open the terminal



Run command python3 test_mymodule.py and this will run the tests.

You should see a screen like this now.



An `OK` in the last line indicates that all tests passed successfully.

`FAILED` in the last line indicates that at least one test has failed, and python prints which test or tests failed.

Write unit tests for the given function

Here is a function that accepts two arguments and returns their sum.

Copy and paste the below code into mymodule.py and the save the file.

```
def add(a,b):
    """
    This function returns the sum of the given numbers
    """
    return a + b
```

- When 2 and 4 are given as input the output must be 6.
- When 0 and 0 are given as input the output must be 0.
- When 2.3 and 3.6 are given as input the output must be 5.9.
- When the strings 'hello' and 'world' are given as input the output must be 'helloworld'.
- When 2.3000 and 4.300 are given as input the output must be 6.6.
- When -2 and -2 are given as input the output must **not** be 0. (Hint : Use assertNotEqual)

Author(s)

Ramesh Sannareddy

Other Contributors

Rav Ahuja

© IBM Corporation. All rights reserved.