

Hands-on Lab: Unit Testing



Unit Testing Lab

Estimated time needed: 30 minutes

Objectives

After completing this lab you will be able to:

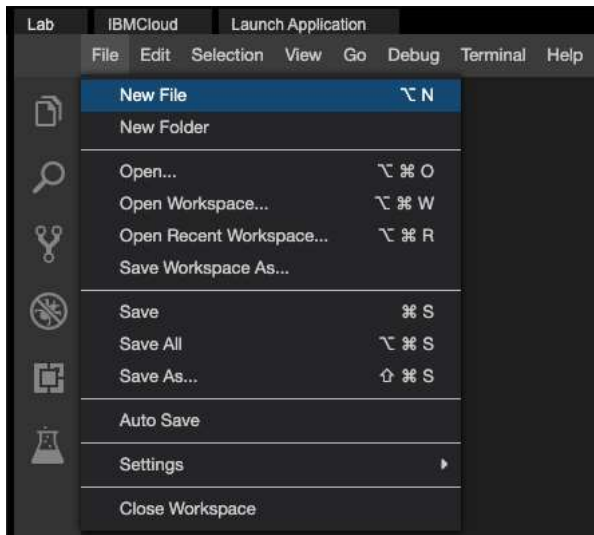
- Write unit tests to test a function.
- Run unit tests and interpret the results.

About the lab environment

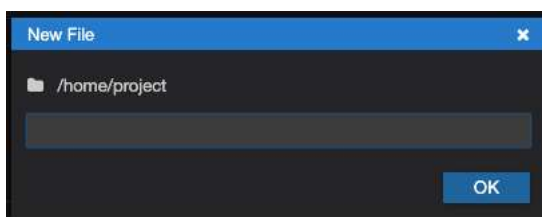
Cloud IDE is an open-source IDE(Integrated Development Environment), that can be run on desktop or on cloud. You will be using the Cloud IDE to do this lab. When you log into the Cloud IDE environment, you are presented with a 'dedicated computer on the cloud' exclusively for you. This is available to 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

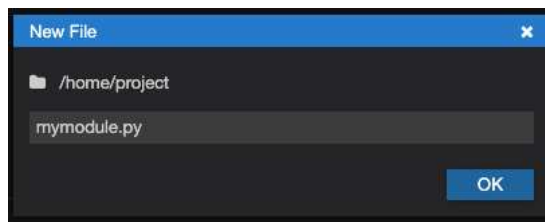
On the window to the right, click on the **File** menu and select **New File** option, as shown in the image below.



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

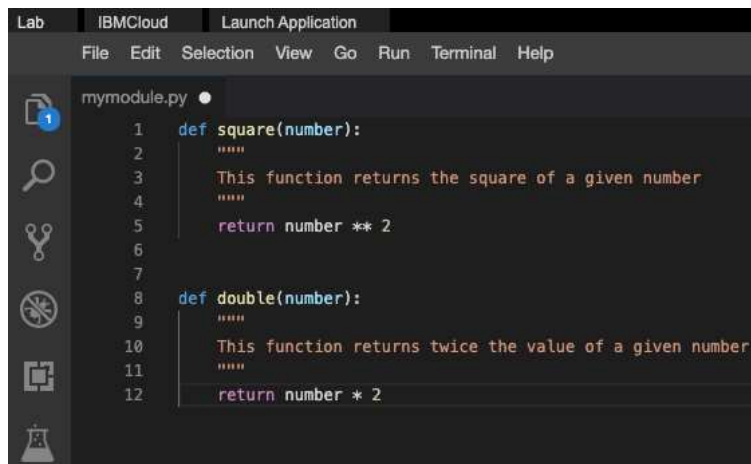
Copy and paste the below code into mymodule.py

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10
11. 11

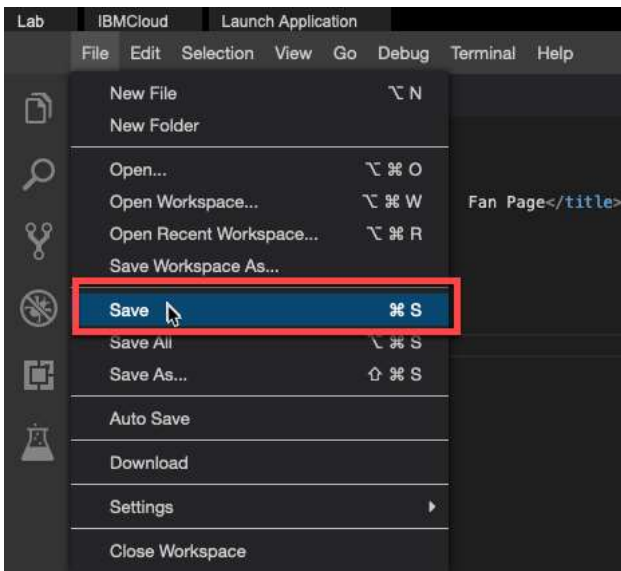
1. def square(number):
2.     """
3.     This function returns the square of a given number
4.     """
5.     return number ** 2
6.
7. def double(number):
8.     """
9.     This function returns twice the value of a given number
10.    """
11.    return number * 2
```

Copied!

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

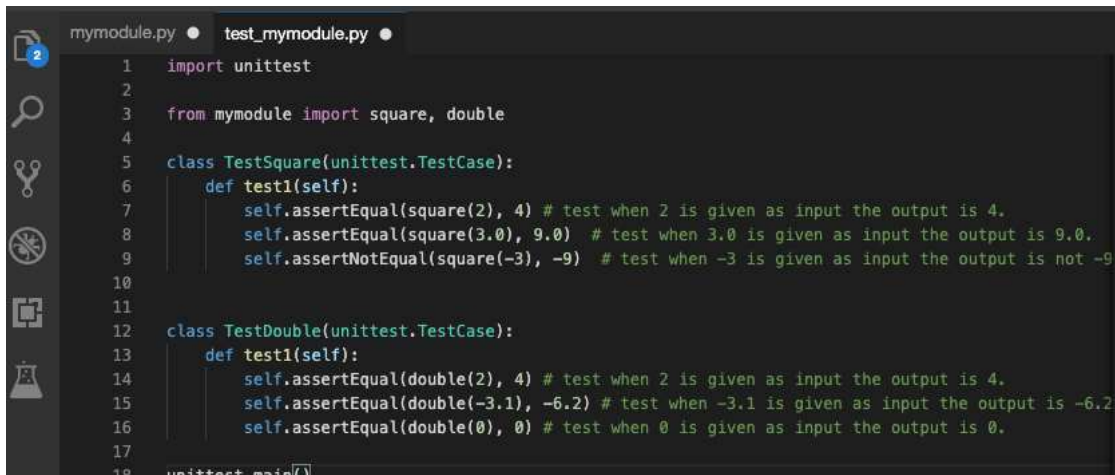
Copy and paste the below code into test_mymodule.py

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10
11. 11
12. 12
13. 13
14. 14
15. 15
16. 16
17. 17
18. 18

1. import unittest
2.
3. from mymodule import square, double
4.
5. class TestSquare(unittest.TestCase):
6.     def test1(self):
7.         self.assertEqual(square(2), 4) # test when 2 is given as input the output is 4.
8.         self.assertEqual(square(3.0), 9.0) # test when 3.0 is given as input the output is 9.0.
9.         self.assertNotEqual(square(-3), -9) # test when -3 is given as input the output is not -9.
10.
11.
12. class TestDouble(unittest.TestCase):
13.     def test1(self):
14.         self.assertEqual(double(2), 4) # test when 2 is given as input the output is 4.
15.         self.assertEqual(double(-3.1), -6.2) # test when -3.1 is given as input the output is -6.2.
16.         self.assertEqual(double(0), 0) # test when 0 is given as input the output is 0.
17.
18. unittest.main()
```

Copied!

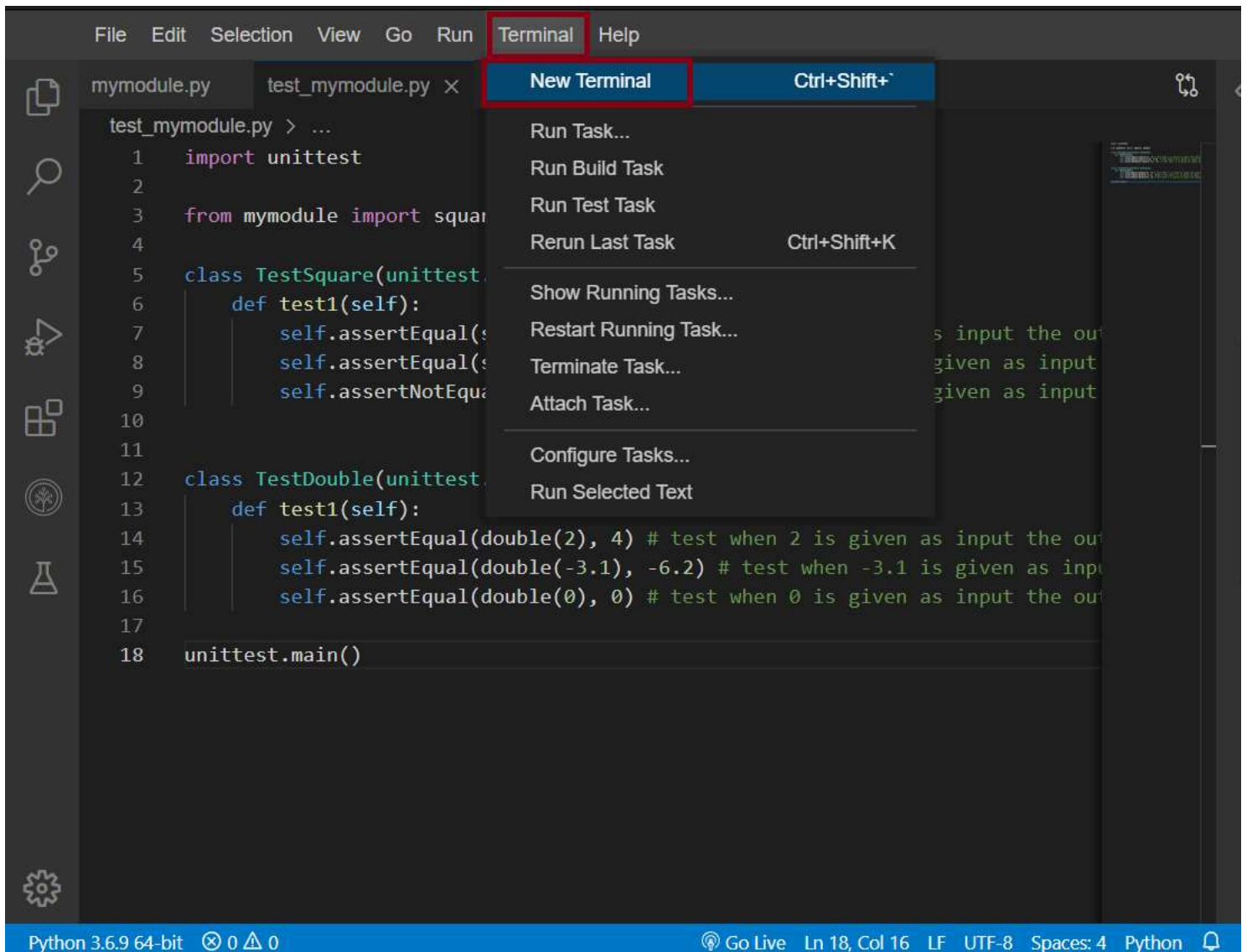
You should see a screen like this now.



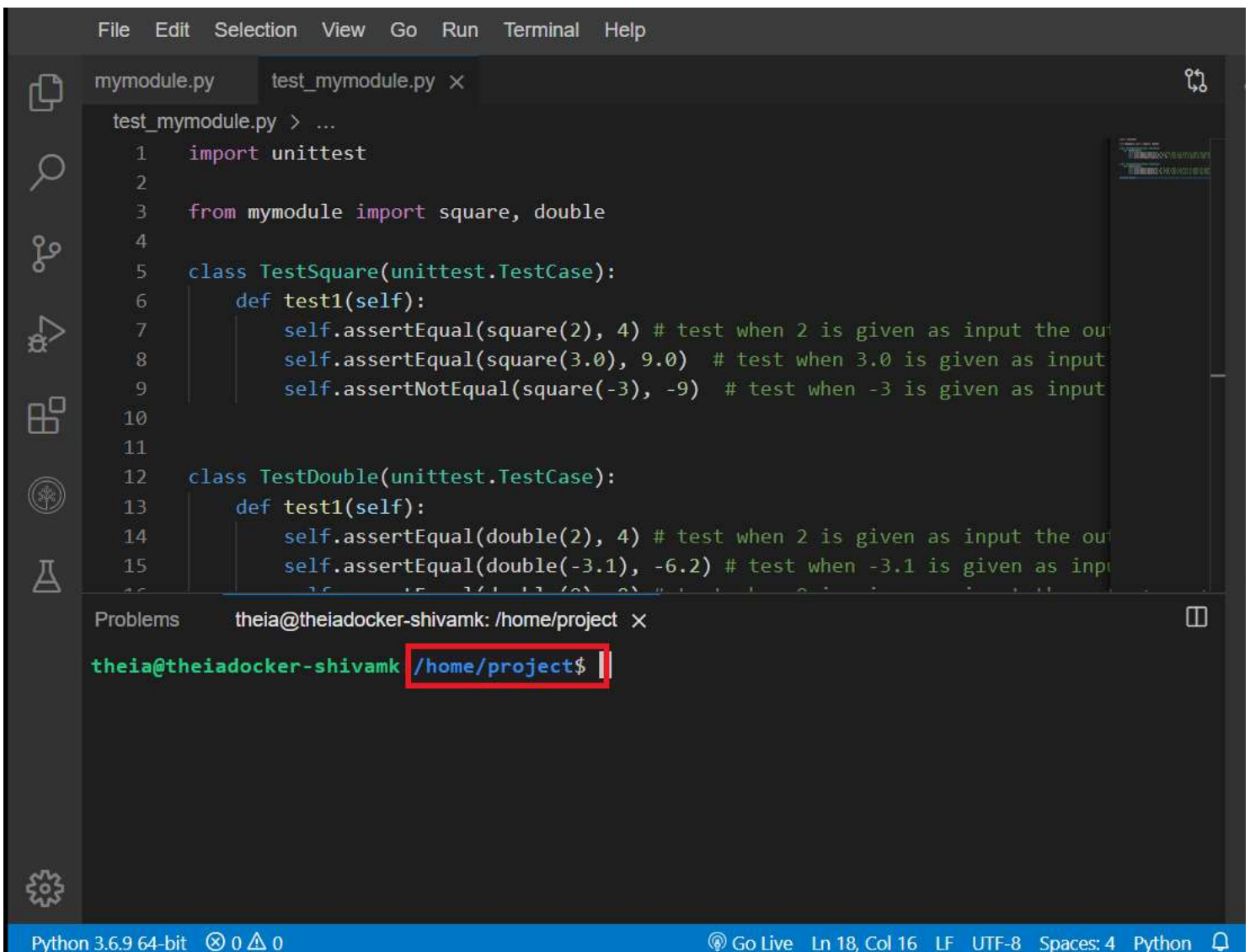
```
1 import unittest
2
3 from mymodule import square, double
4
5 class TestSquare(unittest.TestCase):
6     def test1(self):
7         self.assertEqual(square(2), 4) # test when 2 is given as input the output is 4.
8         self.assertEqual(square(3.0), 9.0) # test when 3.0 is given as input the output is 9.0.
9         self.assertNotEqual(square(-3), -9) # test when -3 is given as input the output is not -9
10
11
12 class TestDouble(unittest.TestCase):
13     def test1(self):
14         self.assertEqual(double(2), 4) # test when 2 is given as input the output is 4.
15         self.assertEqual(double(-3.1), -6.2) # test when -3.1 is given as input the output is -6.2
16         self.assertEqual(double(0), 0) # test when 0 is given as input the output is 0.
17
18 unittest.main()
```

Run tests

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



It will open the terminal



The image shows a screenshot of an IDE with a dark theme. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The editor has two tabs: mymodule.py and test_mymodule.py. The test_mymodule.py file is open, showing a Python script with unittest classes. The script defines two test classes, TestSquare and TestDouble, each with a test1 method. The TestSquare class tests the square function with inputs 2, 3.0, and -3. The TestDouble class tests the double function with inputs 2 and -3.1. The terminal window at the bottom shows the command prompt theia@theiadocker-shivamk: /home/project. The status bar at the bottom indicates Python 3.6.9 64-bit, 0 errors, 0 warnings, and other details like Go Live, Ln 18, Col 16, LF, UTF-8, Spaces: 4, Python.

```
File Edit Selection View Go Run Terminal Help

test_mymodule.py > ...
1  import unittest
2
3  from mymodule import square, double
4
5  class TestSquare(unittest.TestCase):
6      def test1(self):
7          self.assertEqual(square(2), 4) # test when 2 is given as input the out
8          self.assertEqual(square(3.0), 9.0) # test when 3.0 is given as input
9          self.assertNotEqual(square(-3), -9) # test when -3 is given as input
10
11
12  class TestDouble(unittest.TestCase):
13      def test1(self):
14          self.assertEqual(double(2), 4) # test when 2 is given as input the out
15          self.assertEqual(double(-3.1), -6.2) # test when -3.1 is given as input
16
17
18  if __name__ == '__main__':
19      unittest.main()
```

Problems theia@theiadocker-shivamk: /home/project

theia@theiadocker-shivamk /home/project\$

Python 3.6.9 64-bit 0 0 Go Live Ln 18, Col 16 LF UTF-8 Spaces: 4 Python

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

You should see a screen like this now.

The screenshot shows a code editor with two tabs: `mymodule.py` and `test_mymodule.py`. The `test_mymodule.py` file contains the following code:

```
test_mymodule.py > ...
1  import unittest
2
3  from mymodule import square, double
4
5  class TestSquare(unittest.TestCase):
6      def test1(self):
7          self.assertEqual(square(2), 4) # test when 2 is given as input the output should be 4
8          self.assertEqual(square(3.0), 9.0) # test when 3.0 is given as input the output should be 9.0
9          self.assertNotEqual(square(-3), -9) # test when -3 is given as input the output should not be -9
10
11
12  class TestDouble(unittest.TestCase):
13      def test1(self):
14          self.assertEqual(double(2), 4) # test when 2 is given as input the output should be 4
15          self.assertEqual(double(-3.1), -6.2) # test when -3.1 is given as input the output should be -6.2
```

Below the code editor, the `Problems` panel shows the command `python3 test_mymodule.py` being executed. The output is as follows:

```
theia@theiadocker-shivamk:/home/project$ python3 test_mymodule.py
..
-----
Ran 2 tests in 0.000s

OK
theia@theiadocker-shivamk:/home/project$
```

The status bar at the bottom indicates the environment is Python 3.6.9 64-bit, with 0 errors and 0 warnings. The cursor is at line 18, column 16.

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 save the file.

```
1. 1
2. 2
3. 3
4. 4
5. 5

1. def add(a,b):
2.     """
3.     This function returns the sum of the given numbers
4.     """
5.     return a + b
```

Copied!

- 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

Changelog

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2020-11-25	0.1	Ramesh Sannareddy	Created initial version of the lab
2022-10-21	0.2	Shivam Kumar	Updated screenshots

© IBM Corporation 2023. All rights reserved.

[MIT License](#)