**Hands-on Lab: Unit Testing**



**Unit Testing Lab**

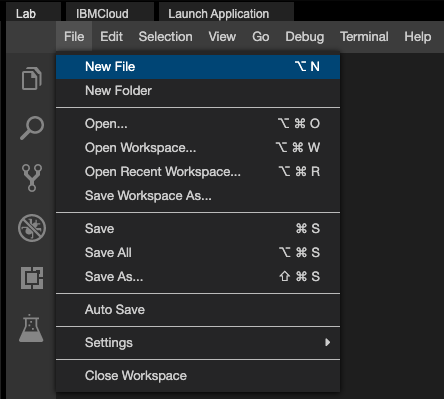
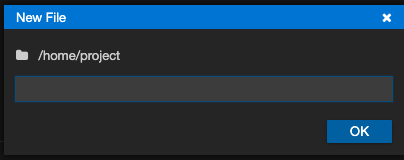
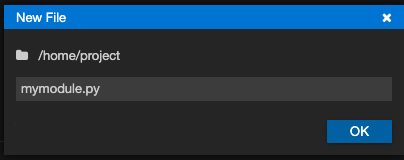
**Estimated time needed: 30 minutes**

**Objectives**

After completing this lab you will be able to:

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

**reate 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.

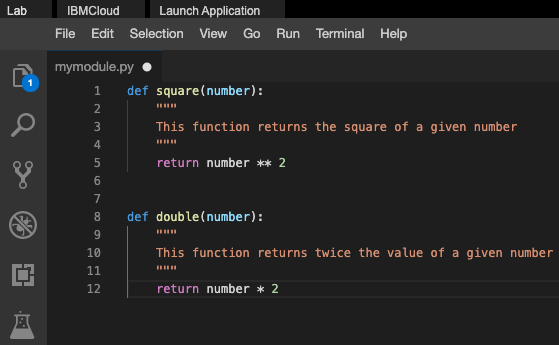
You are now ready to add code to mymodule.py

Copy and paste the below code into mymodule.py

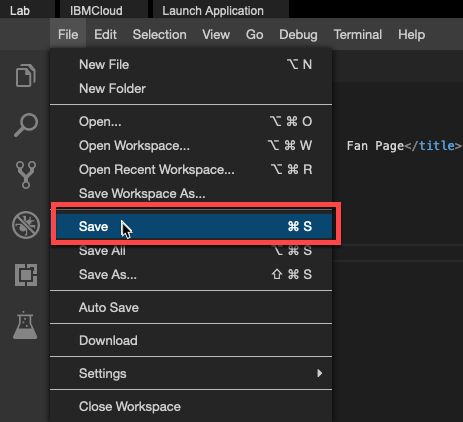
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10
11. 11
12. def square(number):
13. """
14. This function returns the square of a given number
15. """
16. return number \*\* 2
17. def double(number):
18. """
19. This function returns twice the value of a given number
20. """
21. return number \* 2

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
19. import unittest
20. from mymodule import square, double
21. class TestSquare(unittest.TestCase):
22. def test1(self):
23. self.assertEqual(square(2), 4) # test when 2 is given as input the output is 4.
24. self.assertEqual(square(3.0), 9.0) # test when 3.0 is given as input the output is 9.0.
25. self.assertNotEqual(square(-3), -9) # test when -3 is given as input the output is not -9.
27. class TestDouble(unittest.TestCase):
28. def test1(self):
29. self.assertEqual(double(2), 4) # test when 2 is given as input the output is 4.
30. self.assertEqual(double(-3.1), -6.2) # test when -3.1 is given as input the output is -6.2.
31. self.assertEqual(double(0), 0) # test when 0 is given as input the output is 0.
33. unittest.main()

Copied!

You should see a screen like this now.

A screenshot of a computer program

Description automatically generated

**Hands-on Lab: Creating a Python Package**



**Creating a Python Package**

Estimated time needed: **30** minutes

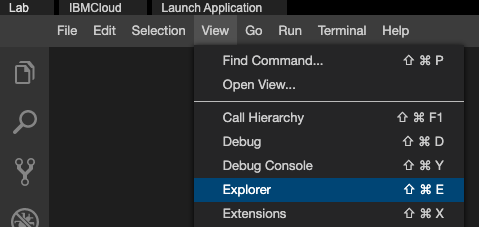
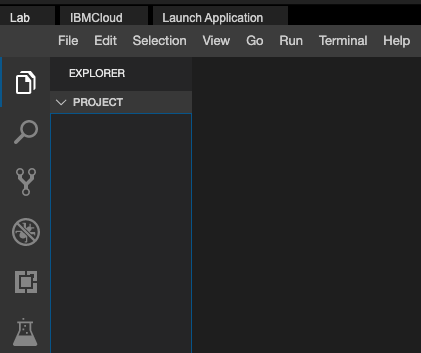
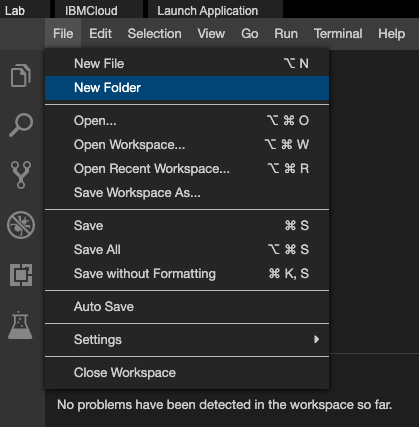
**Objectives**

In this lab you will :

* Create a module named basic
* Add two functions to the module basic
* Create a module named stats
* Add two functions to the module stats
* Create a python package named mymath
* Verify that the package is working

# Lab

### Create Package

* On the window to the right, click on the **View** menu and select **Explorer** option, as shown in the image below.  
    
    
    
  
* Your IDE now should look like the image below.  
    
    
    
  
* On the window to the right, click on the **File** menu and select **New Folder** option, as shown in the image below.  
    
    
    
  
* Enter **mymath** and click OK as shown in the image below.  
    
    
    
  A black and blue rectangular object with a black border

  Description automatically generated

**Create the first module**

* Create a python module named basic

Create a file named **basic.py**.

Copy and paste the below code into basic.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. def square(number):
19. """
20. This function returns the square of a given number
21. """
22. return number \*\* 2
23. def double(number):
24. """
25. This function returns twice the value of a given number
26. """
27. return number \* 2
28. def add(a, b):
29. """
30. This function returns the sum of given numbers
31. """
32. return a + b

Copied!

You should see a screen like this now.  
A computer screen shot of a program

Description automatically generated

Save the file **basic.py**

**Create the second module**

* Create a module named stats

Create a file named **stats.py**.

Copy and paste the below code into stats.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
19. 19
20. def mean(numbers):
21. """
22. This function returns the mean of the given list of numbers
23. """
24. return sum(numbers)/len(numbers)
25. def median(numbers):
26. """
27. This function returns median of the given list of numbers
28. """
29. numbers.sort()
31. if len(numbers) % 2 == 0:
32. median1 = numbers[len(numbers) // 2]
33. median2 = numbers[len(numbers) // 2 - 1]
34. mymedian = (median1 + median2) / 2
35. else:
36. mymedian = numbers[len(numbers) // 2]
37. return mymedian

Copied!

You should see a screen like this now.  
A computer screen shot of a program

Description automatically generated

Save the file **stats.py**

**Create init.py**

* Create the file \_\_init\_\_.py

Copy and paste the below code into \_\_init\_\_.py

1. 1
2. 2
3. from . import basic
4. from . import stats

Copied!

Save the file \_\_init\_\_.py

Now your directory structure should look like

1. 1
2. 2
3. 3
4. 4
5. mymath
6. mymath/\_\_init\_\_.py
7. mymath/basic.py
8. mymath/statistics.py

Copied!

A screenshot of a computer program

Description automatically generated

You are done creating a package

**Create init.py**

* Create the file \_\_init\_\_.py

Copy and paste the below code into \_\_init\_\_.py

1. 1
2. 2
3. from . import basic
4. from . import stats

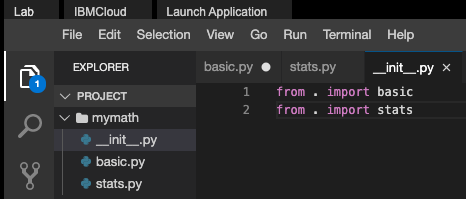
Copied!

Save the file \_\_init\_\_.py

Now your directory structure should look like

1. 1
2. 2
3. 3
4. 4
5. mymath
6. mymath/\_\_init\_\_.py
7. mymath/basic.py
8. mymath/statistics.py

Copied!



You are done creating a package

**Verify the package**

* On the window to the right, click on the **Terminal** menu and select **New Terminal** option, as shown in the image below.
* You will see a terminal open up on the bottom of the screen like the one in the image below.  
    
  A screen shot of a computer

  Description automatically generated
* At the terminal type **python3** to invoke python interpreter.
* Once the python interpreter is loaded.
* At the python prompt type **import mymath**
* If the above command runs without errors, it is an indication that the mymath package is successfully loaded.
* At the python prompt type **mymath.basic.add(3,4)**
* You should see an output *7* on the screen.
* At the python prompt type **mymath.stats.mean([3,4,5])**
* You should see an output *4.0* on the screen.
* Type **exit()** to quit python interpreter.

A screen shot of a computer program

Description automatically generated

# Practice Exercise

### Create a new module named geometry and add to the mymath package.

* Create a module name geometry
* Add a function named area\_of\_rectangle that takes length and breadth as input and returns the area of a rectangle.
* Add a function named area\_of\_circle that takes radius as input and returns the area of a circle.
* Modify the \_\_init\_\_.py to include this module.
* Import and test the function area\_of\_circle from python terminal.

## Authors

Ramesh Sannareddy

### Other Contributors

Rav Ahuja

## Change Log

| **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 | 1.0 | Ratima | Updated Skill Network Logo screenshot |

Copyright © 2020 IBM Corporation. This notebook and its source code are released under the terms of the [MIT License](https://cognitiveclass.ai/mit-license?cm_mmc=Email_Newsletter-_-Developer_Ed%2BTech-_-WW_WW-_-SkillsNetwork-Courses-IBM-DA0321EN-SkillsNetwork-21426264&cm_mmca1=000026UJ&cm_mmca2=10006555&cm_mmca3=M12345678&cvosrc=email.Newsletter.M12345678&cvo_campaign=000026UJ&cm_mmc=Email_Newsletter-_-Developer_Ed%2BTech-_-WW_WW-_-SkillsNetwork-Courses-IBM-DA0321EN-SkillsNetwork-21426264&cm_mmca1=000026UJ&cm_mmca2=10006555&cm_mmca3=M12345678&cvosrc=email.Newsletter.M12345678&cvo_campaign=000026UJ&cm_mmc=Email_Newsletter-_-Developer_Ed%2BTech-_-WW_WW-_-SkillsNetwork-Courses-IBM-DA0321EN-SkillsNetwork-21426264&cm_mmca1=000026UJ&cm_mmca2=10006555&cm_mmca3=M12345678&cvosrc=email.Newsletter.M12345678&cvo_campaign=000026UJ&cm_mmc=Email_Newsletter-_-Developer_Ed%2BTech-_-WW_WW-_-SkillsNetwork-Courses-IBM-DA0321EN-SkillsNetwork-21426264&cm_mmca1=000026UJ&cm_mmca2=10006555&cm_mmca3=M12345678&cvosrc=email.Newsletter.M12345678&cvo_campaign=000026UJ).

# Cheatsheet: Python Coding Practices and Packaging Concepts

A logo with text on it

Description automatically generated

**Estimated time needed:** 5 minutes

| **Package/Method** | **Description** | **Code Example** |
| --- | --- | --- |
| **Packaging** | To create a package, the folder structure is as follows:   1. **Project folder** → **Package name** → **init.py**, **module\_1.py**, **module\_2.py**, and so on. 2. In the **init.py** file, add code to reference the modules in the package. | **module1.py**  def function\_1(arg):      return <operation output>  **init.py**:  from . import function\_1 |
| **Python Style Guide** | * Four spaces for indentation * Use blank lines to separate functions and classes * Use spaces around operators and after commas * Add larger blocks of code inside functions * Name functions and files using lowercase with underscores * Name classes using CamelCase * Name constants in capital letters with underscores separating words | def function\_1(a, b):       if a > b:           c = c + 5      else:          c = c - 3      return c  …  c = function\_1(a, b)  **Constant Definition example**  MAX\_FILE\_UPLOAD\_SIZE |
| **Static Code Analysis** | Use Static code analysis method to evaluate your code against a predefined style and standard without executing the code.  For example, use Pylint to perform static code analysis. | **Shell code:**  pylint code.py |
| **Unit Testing** | Unit testing is a method to validate if units of code are operating as designed.  During code development, each unit is tested. The unit is tested in a continuous integration/continuous delivery test server environment.  You can use different test functions to build unit tests and review the unit test output to determine if the test passed or failed. | import unittest  from mypackage.mymodule import my\_function  class TestMyFunction(unittest.TestCase):      def test\_my\_function(self):          result = my\_function(<args>)          self.asserEqual(result, <response>)  unittest.main() |

## Author(s)

Abhishek Gagneja

### Other Contributor(s)

Andrew Pfeiffer, Sina Nazeri

## Changelog

| **Date** | **Version** | **Changed by** | **Change Description** |
| --- | --- | --- | --- |
| 2023-07-06 | 0.1 | Abhishek Gagneja | Initial version created |
| 2023-07-10 | 0.2 | Andrew Pfeiffer | Documented in AWB |