

# LAB ASSIGNMENT-8.4

Name: P. Umesh Reddy

Hall-Ticket No: 2403a510f9

Batch: 06

Course: AI Assisted Coding

## Task 1

### Task Description#1

- Write a test case to check if a function returns the square of a number.
- Then write the function with help from GitHub Copilot or Cursor AI.

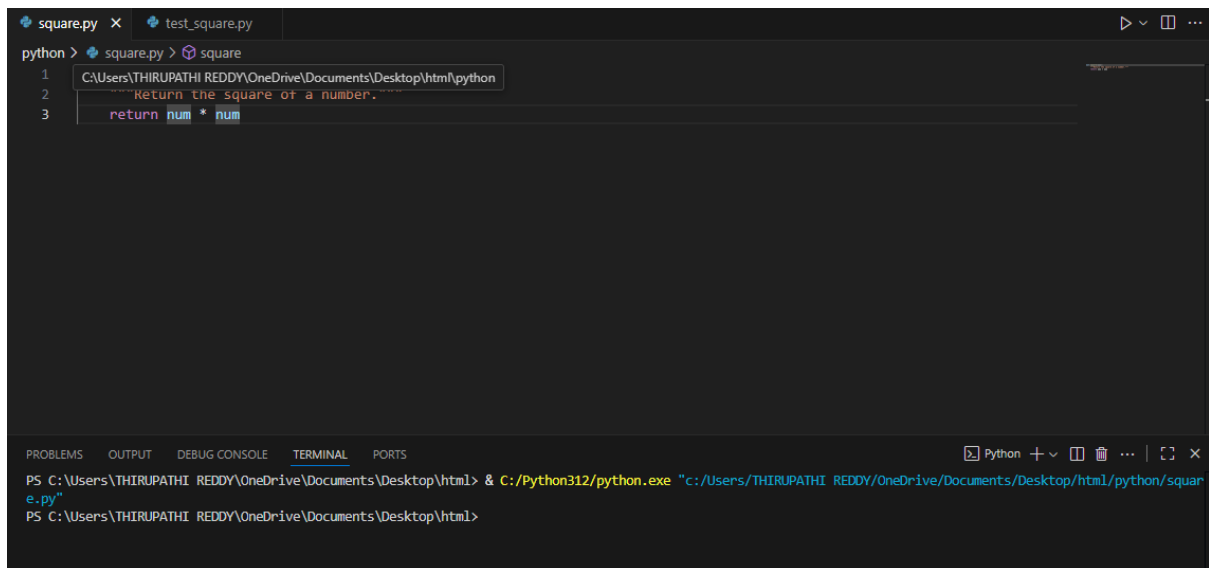
### Expected Outcome#1

- A test file and function file with passing test cases and working logic

### #Prompt:

- A test file and function file with passing test cases and working logic

### Function File and Output:



The screenshot shows a code editor with two files: `square.py` and `test_square.py`. The `square.py` file contains a Python function `square` that takes a number `num` as input and returns its square by multiplying it by itself. The function is defined as follows:

```
1 def square(num):  
2     """Return the square of a number."""  
3     return num * num
```

The `test_square.py` file contains test cases for the `square` function. The test cases are as follows:

```
1 import unittest  
2 from square import square  
3  
4 class TestSquare(unittest.TestCase):  
5     def test_square(self):  
6         self.assertEqual(square(4), 16)  
7         self.assertEqual(square(9), 81)  
8         self.assertEqual(square(16), 256)  
9  
10 if __name__ == '__main__':  
11     unittest.main()
```

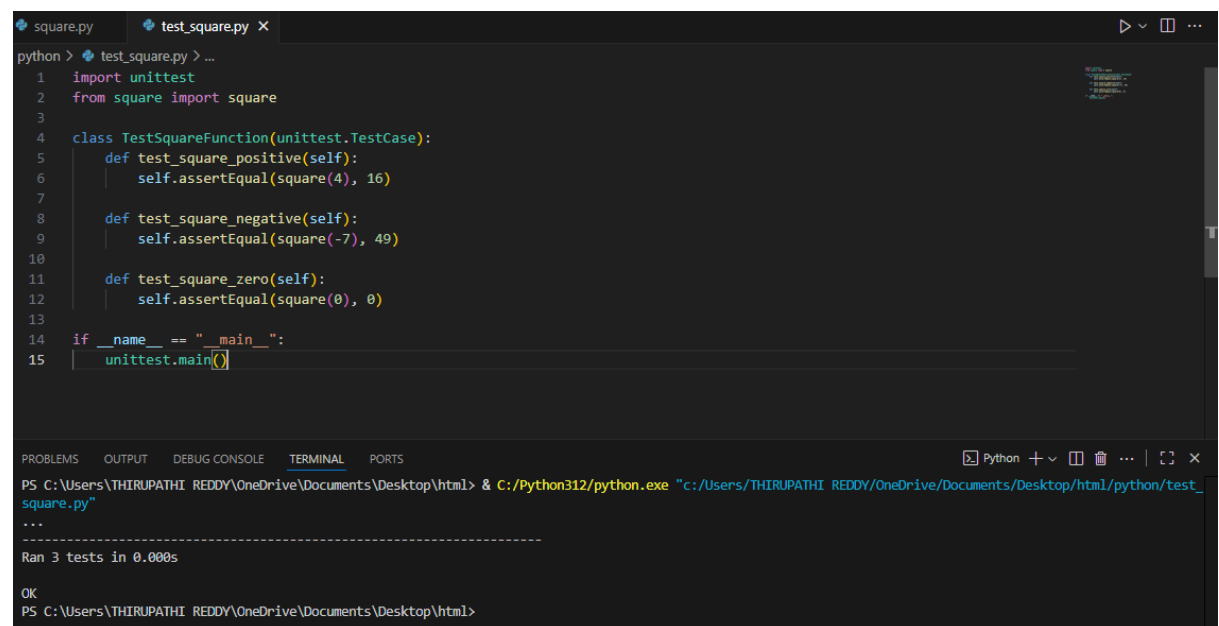
The terminal output shows the command to run the test cases:

```
PS C:\Users\THIRUPATHI REDDY\OneDrive\Documents\Desktop\html> & C:/Python312/python.exe "c:/Users/THIRUPATHI REDDY/OneDrive/Documents/Desktop/html/python/square.py"  
PS C:\Users\THIRUPATHI REDDY\OneDrive\Documents\Desktop\html>
```

### Code Explanation:

- The `square` function takes a number as input and returns its square by multiplying the number by itself.
- The docstring describes the function's purpose.
- This function will pass all the test cases in your `test_square.py` file.

## Test File & Output:



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

```
python > test_square.py > ...
1 import unittest
2 from square import square
3
4 class TestSquareFunction(unittest.TestCase):
5     def test_square_positive(self):
6         self.assertEqual(square(4), 16)
7
8     def test_square_negative(self):
9         self.assertEqual(square(-7), 49)
10
11    def test_square_zero(self):
12        self.assertEqual(square(0), 0)
13
14    if __name__ == "__main__":
15        unittest.main()
```

The terminal output at the bottom shows the command executed and the result:

```
PS C:\Users\THIRUPATHI REDDY\OneDrive\Documents\Desktop\html> & C:/Python312/python.exe "c:/Users/THIRUPATHI REDDY/OneDrive/Documents/Desktop/html/python/test_square.py"
...
-----
Ran 3 tests in 0.000s
OK
PS C:\Users\THIRUPATHI REDDY\OneDrive\Documents\Desktop\html>
```

## Code Explanation:

- **Imports:**
  - [import unittest](#) imports Python's built-in unit testing framework.
  - [from square import square](#) imports the [square](#) function from the [square.py](#) file.
- **Test Class:**
  - [class TestSquareFunction\(unittest.TestCase\):](#) defines a test case class that inherits from [unittest.TestCase](#).
- **Test Methods:**
  - [test\\_square\\_positive](#): Checks if [square\(4\)](#) returns 16.
  - [test\\_square\\_negative](#): Checks if [square\(-7\)](#) returns 49.
  - [test\\_square\\_zero](#): Checks if [square\(0\)](#) returns 0.
- **Assertions:**
  - Each test uses [self.assertEqual\(\)](#) to compare the function's output to the expected result.
- **Main Block:**
  - [if \\_\\_name\\_\\_ == "\\_\\_main\\_\\_": unittest.main\(\)](#) runs all test cases when the script is executed directly.

## Comments:

This test file verifies that the [square](#) function works correctly for positive numbers, negative numbers, and zero. All tests should pass if the function is implemented properly.

## Task 2

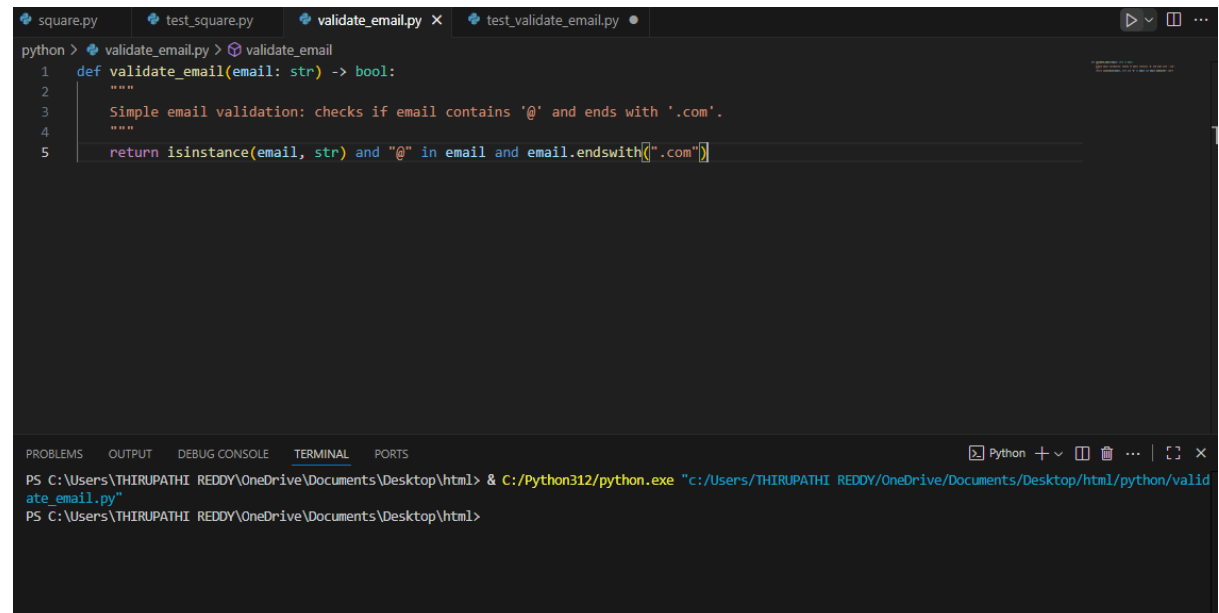
### Task Description#2

- Create test cases to validate an email address (e.g., contains @ and .com).
- Use AI assistance to implement the validate\_email() function.

### Expected Outcome#2

- Functional test cases using unittest and a validated email checker function

### Code & Output:



```
python > validate_email.py > validate_email
1 def validate_email(email: str) -> bool:
2     """
3     Simple email validation: checks if email contains '@' and ends with '.com'.
4     """
5     return isinstance(email, str) and "@" in email and email.endswith(".com")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\THIRUPATHI REDDY\OneDrive\Documents\Desktop\html> & C:/Python312/python.exe "c:/Users/THIRUPATHI REDDY/OneDrive/Documents/Desktop/html/python/validate_email.py"
PS C:\Users\THIRUPATHI REDDY\OneDrive\Documents\Desktop\html>
```

### Ode Explanation:

- The function [validate\\_email](#) takes an email string as input.
- It returns True only if:
  - The input is a string,
  - It contains the '@' symbol,
  - It ends with '.com'.
- Otherwise, it returns False.
- The docstring describes the function's purpose.

Code:

```
python > test_validate_email.py > ...
1 import unittest
2 from validate_email import validate_email
3
4 class TestValidateEmail(unittest.TestCase):
5     def test_valid_email(self):
6         self.assertTrue(validate_email("user@example.com"))
7
8     def test_missing_at_symbol(self):
9         self.assertFalse(validate_email("userexample.com"))
10
11    def test_missing_dotcom(self):
12        self.assertFalse(validate_email("user@example.org"))
13
14    def test_empty_string(self):
15        self.assertFalse(validate_email(""))
16
17    def test_non_string_input(self):
18        self.assertFalse(validate_email(12345))
19
20    def test_multiple_at_symbols(self):
21        self.assertTrue(validate_email("user@domain@company.com"))
22
23 if __name__ == "__main__":
24     unittest.main()
```

Output:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + - [ ] [ ] ... [ ] [ ] X
PS C:\Users\THIRUPATHI REDDY\OneDrive\Documents\Desktop\html> & C:/Python312/python.exe "c:/Users/THIRUPATHI REDDY/OneDrive/Documents/Desktop/html/python/test_validate_email.py"
PS C:\Users\THIRUPATHI REDDY\OneDrive\Documents\Desktop\html> & C:/Python312/python.exe "c:/Users/THIRUPATHI REDDY/OneDrive/Documents/Desktop/html/python/test_validate_email.py"
.....
-----
.....
-----
Ran 6 tests in 0.001s
Ran 6 tests in 0.001s

OK
PS C:\Users\THIRUPATHI REDDY\OneDrive\Documents\Desktop\html>

```

### Code Explanation:

- Uses Python's unittest framework to test the [validate\\_email](#) function.
- Each method checks a different scenario:
  - Valid email,
  - Missing '@',
  - Missing '.com',
  - Empty string,
  - Non-string input,
  - Multiple '@' symbols.
- The tests ensure the function works as expected for various inputs.

Comments: