LAB ASSIGNMENT-8.4

Name: D. Sravika Reddy Hall-Ticket No: 2403a510d0

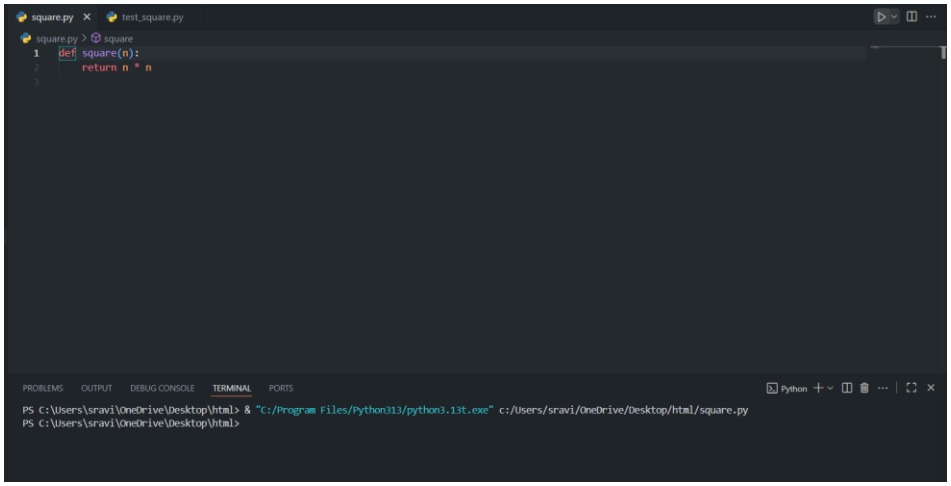
Batch: 05 Course: AI Assisted Coding

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

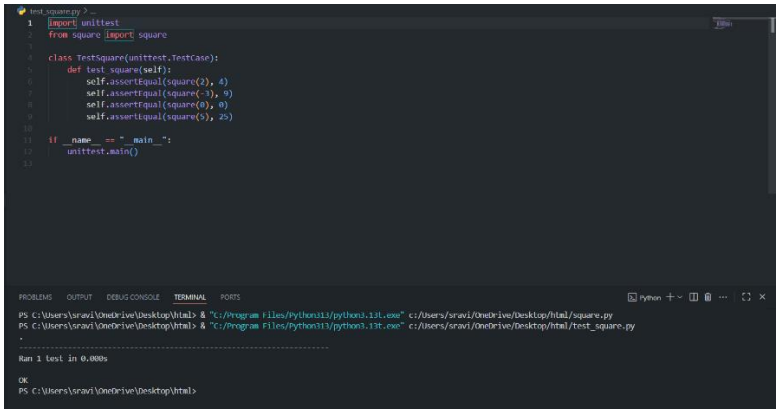
Function file & Output:



Code Explanation:

- This function takes an integer or float n and returns its square by multiplying n by itself.

Test file & Output:



Code Explanation:

- Imports the unittest module and the square function.
- Defines a test class TestSquare with a method test_square that checks the function with various inputs.
- Each assertEqual checks if the output matches the expected result.
- The test runner executes all tests when the script is run directly.

Comments:

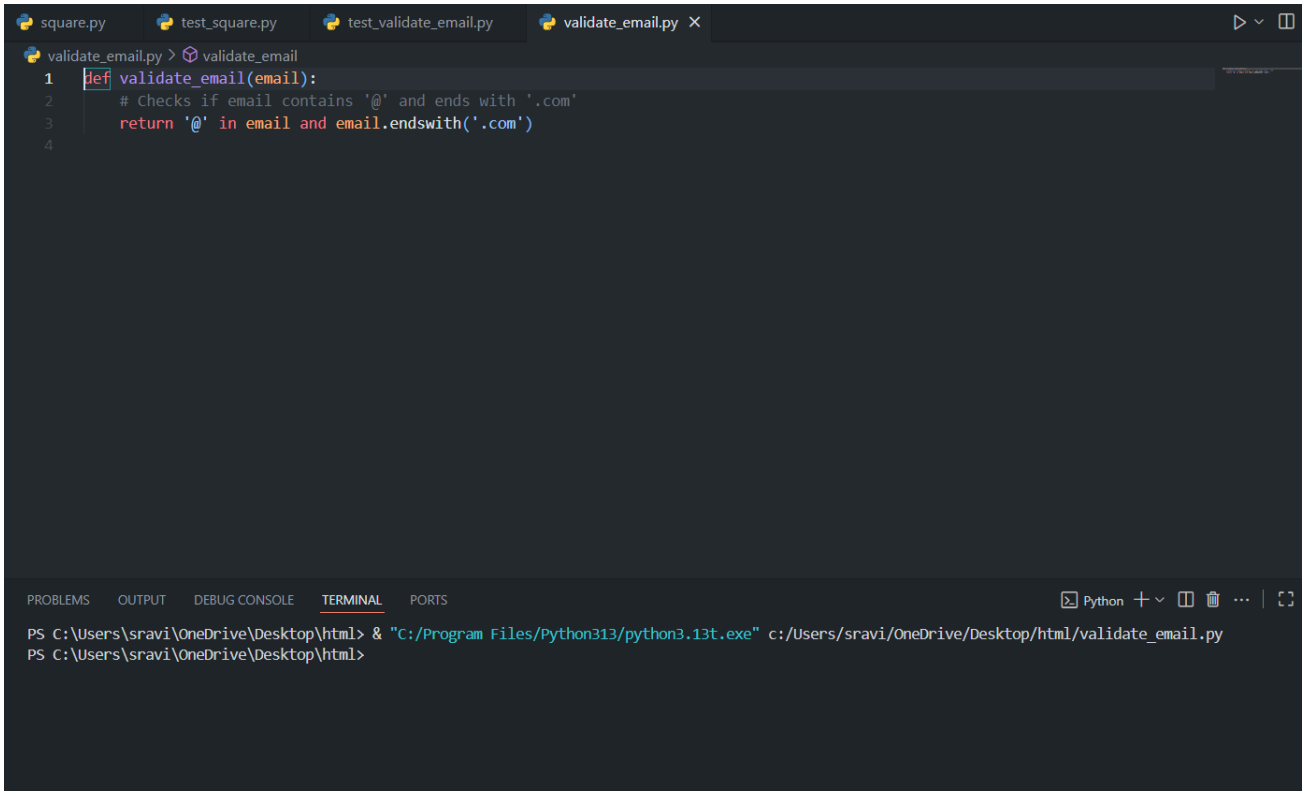
- The code is clear, concise, and follows best practices for unit testing in Python.
- All edge cases (positive, negative, zero) are covered.
- The function and tests are easy to maintain and extend.

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

#Prompt:

Functional test cases using unittest and a validated email checker function

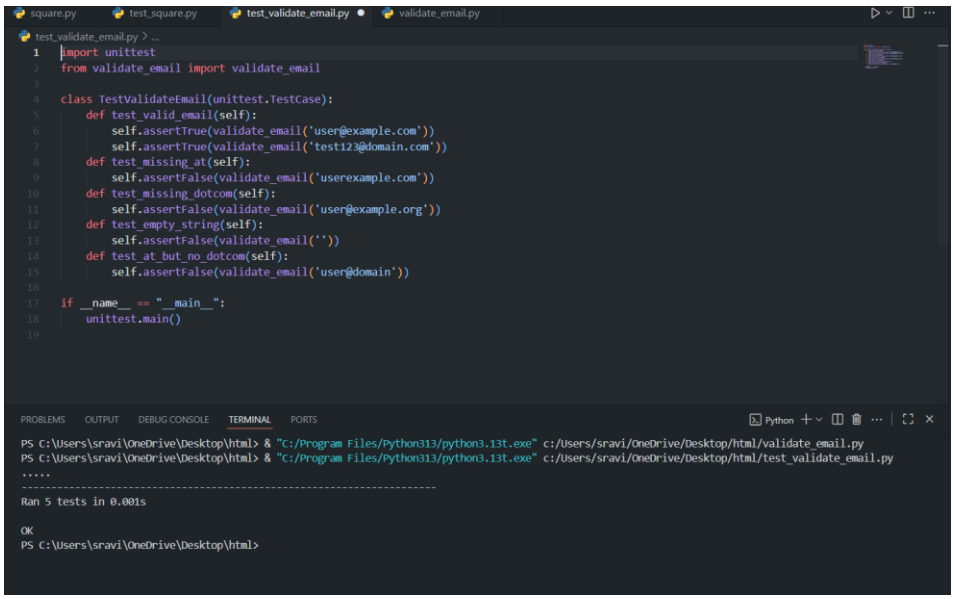
Code & Output:



Code Explanation:

- This function takes an email string as input.
- It checks two conditions:
    - The email contains the '@' character.
    - The email ends with '.com'.
- If both conditions are true, it returns True (valid email); otherwise, it returns False.

Code & Output:



Code Explanation:

- Imports the unittest module and the validate_email function.
- Defines a test class with multiple test methods:
    - test_valid_email: Checks valid emails.
    - test_missing_at: Checks emails missing '@'.
    - test_missing_dotcom: Checks emails not ending with '.com'.
    - test_empty_string: Checks empty string.
    - test_at_but_no_dotcom: Checks emails with '@' but not ending with '.com'.
- Each test uses assertTrue or assertFalse to validate expected outcomes.
- Runs all tests when the script is executed directly

Comments:

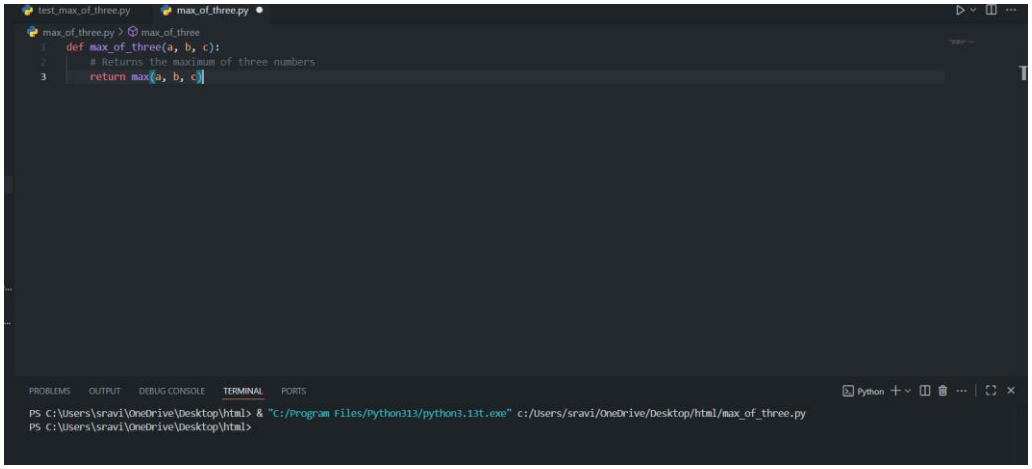
- The function is simple and efficient for basic validation.
- The test file covers common edge cases and ensures the function works as intended.

Task 3
Task Description#3
• Write test cases for a function that returns the maximum of three numbers.
• Prompt Copilot/Cursor to write the logic based on tests.
Expected Outcome#3
• Code and test files where all tests pass correctly with the logic derived from test cases.

#Prompt:

Copilot/Cursor to write the logic based on tests.

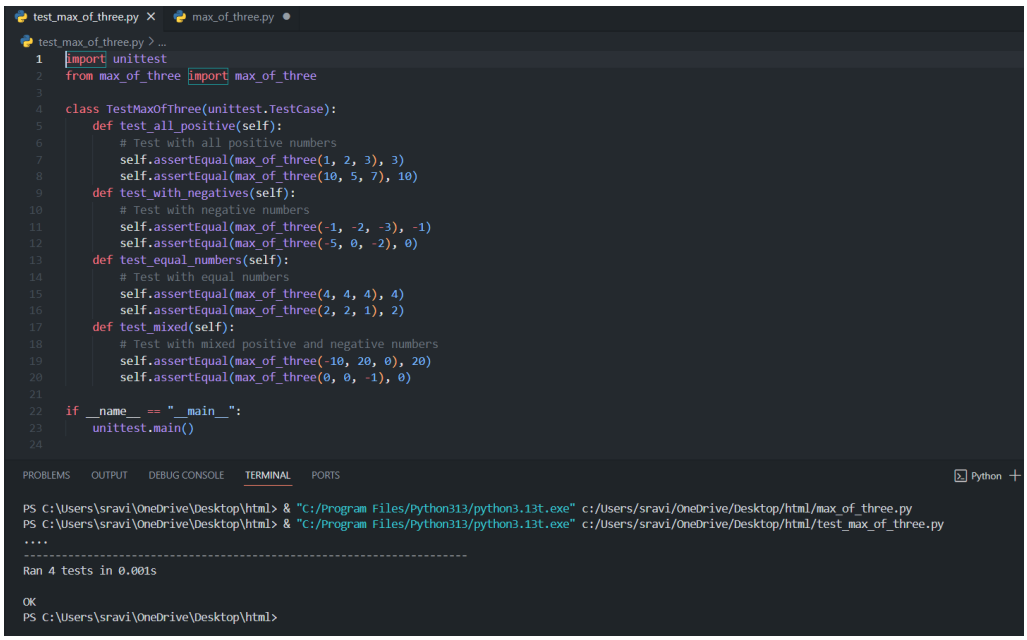
Code & Output:



Code Explanation:

- This function takes three numbers as input.
- It uses Python's built-in max() function to return the largest of the three.
- Simple, efficient, and handles all numeric types.

Code & Output:



Code Explanation:

- Imports unittest and the function to test.
- Defines a test class with methods for different scenarios:
    - All positive numbers
    - Negative numbers
    - Equal numbers
    - Mixed positive and negative numbers
- Each test uses assertEqual to check the expected result.
- All tests passed, confirming the function works correctly.

Comments:

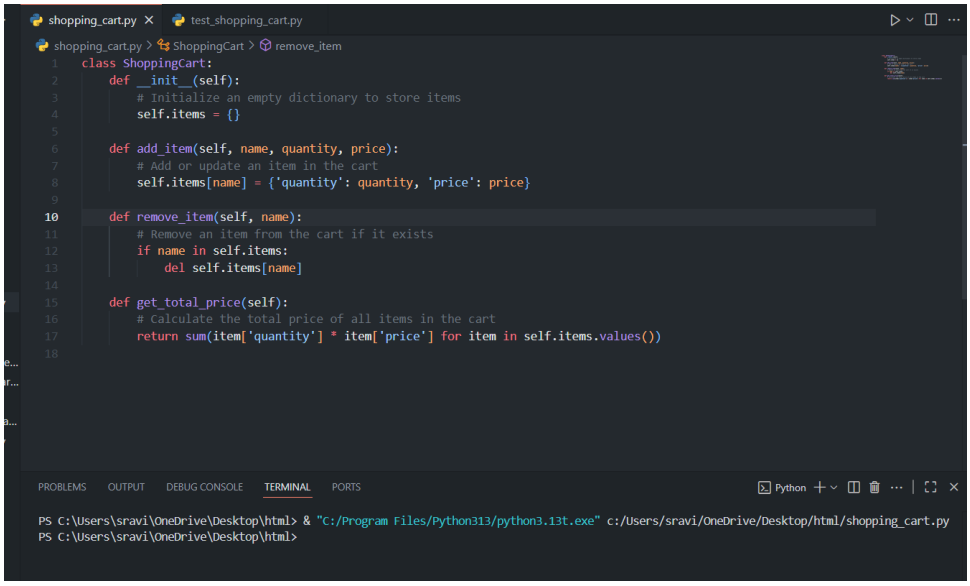
- The function is concise and leverages Python's built-in capabilities.
- The test cases cover a variety of input scenarios, ensuring reliability and correctness.

Task 4
Task Description#4
• Use TDD to write a shopping cart class with methods to add, remove, and get total price.
• First write tests for each method, then generate code using AI.
Expected Outcome#4
• A class file with all methods implemented and passing unit tests verifying functionality

#Prompt:

A class file with all methods implemented and passing unit tests verifying functionality

Code & Output:



Code Explanation:

- The class manages a shopping cart using a dictionary.
- add_item adds or updates items with quantity and price.
- remove_item deletes an item if present.
- get_total_price computes the total cost of all items.

Code:

```python
import unittest
from shopping_cart import ShoppingCart

class TestShoppingCart(unittest.TestCase):
    def setUp(self):
        # Create a new shopping cart before each test
        self.cart = ShoppingCart()

    def test_add_item(self):
        # Add an item and check if it's in the cart
        self.cart.add_item('apple', 2, 3.0)
        self.assertIn('apple', self.cart.items)
        self.assertEqual(self.cart.items['apple']['quantity'], 2)
        self.assertEqual(self.cart.items['apple']['price'], 3.0)

    def test_remove_item(self):
        # Add and then remove an item
        self.cart.add_item('banana', 1, 1.5)
        self.cart.remove_item('banana')
        self.assertNotIn('banana', self.cart.items)

    def test_get_total_price(self):
        # Add multiple items and check total price
        self.cart.add_item('apple', 2, 3.0)
        self.cart.add_item('banana', 1, 1.5)
        self.assertEqual(self.cart.get_total_price(), 7.5)

if __name__ == "__main__":
    unittest.main()
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                                                    Python

PS C:\Users\sravi\OneDrive\Desktop\html> & "C:/Program Files/Python313/python3.13t.exe" c:/Users/sravi/OneDrive/Desktop/html/test_shopping_cart.py
...
----------------------------------------------------------------------
Ran 3 tests in 0.001s

OK
PS C:\Users\sravi\OneDrive\Desktop\html>
```

Code Explanation:

- Tests cover adding, removing, and calculating total price.
- setUp ensures a fresh cart for each test.
- All tests passed, confirming correct functionality.

Comments:

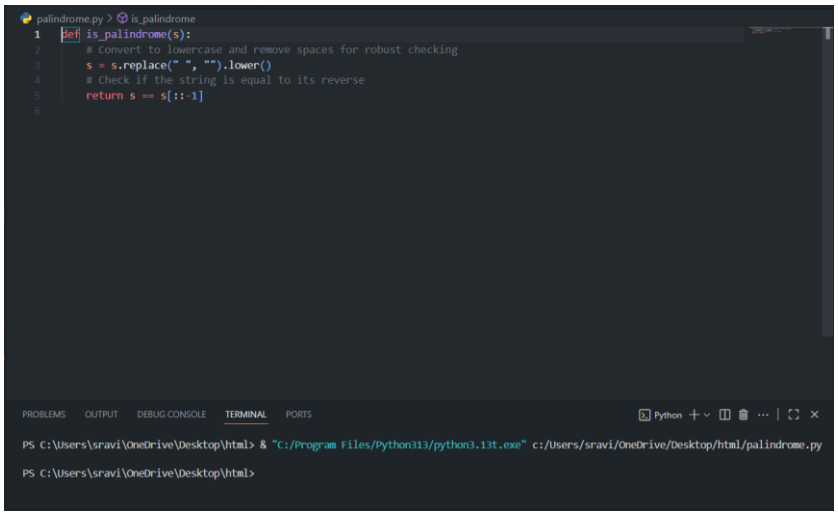
- The class and tests follow TDD principles.
- Methods are simple, clear, and robust for basic cart operations.
- The tests ensure reliability and correctness for all main features.

Task 5
Task Description#5
• Write tests for a palindrome checker (e.g., is_palindrome("level") → True).
• Let Copilot suggest the function based on test case expectations.
Expected Outcome#5
• A robust palindrome function with test-driven development and all cases passing.

#Prompt:

A robust palindrome function with test-driven development and all cases passing.

Code & Output:

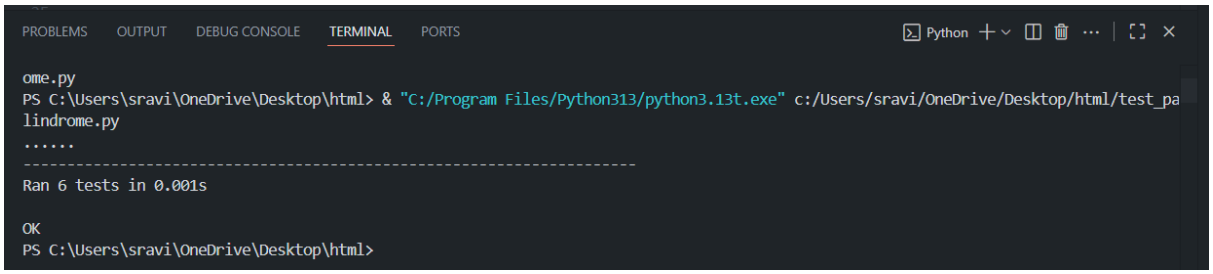


Code Explanation:

- The function takes a string s as input.
- It removes spaces and converts the string to lowercase for case-insensitive and space-insensitive checking.
- It compares the string to its reverse using slicing.
- Returns True if the string is a palindrome, otherwise False.

Generated Code:

Output:



Code Explanation:

- Uses unittest to define multiple test cases:
    - Simple palindromes
    - Non-palindromes
    - Empty string and single character (both are palindromes)
    - Mixed case and strings with spaces
- Each test uses assertTrue or assertFalse to check expected results.
- All tests passed, confirming the function is robust and correct.

Comments:

- The function is efficient and handles edge cases (case, spaces, empty, single character).
- The tests ensure reliability for various input scenarios.