AI Assisted Coding

Assignment 8.3

Name: R.Bharadwaj

Hall ticket no: 2303A51610

Batch no: 19

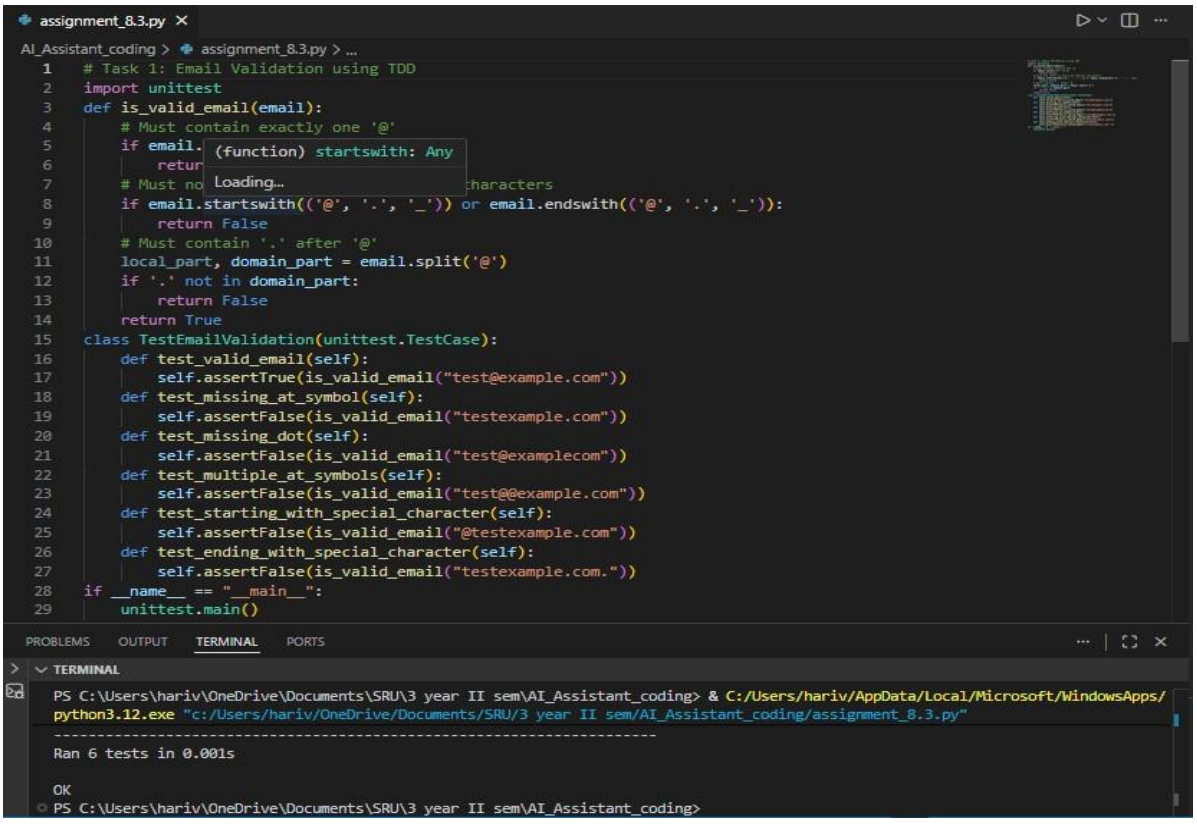
Task 1:

Prompt:

Generate Python unittest test cases for validating email formats based on the following rules:

- Must contain @ and .
- Must not start or end with special characters
- Must not contain multiple @ symbols

Code & Output:



Explanation:

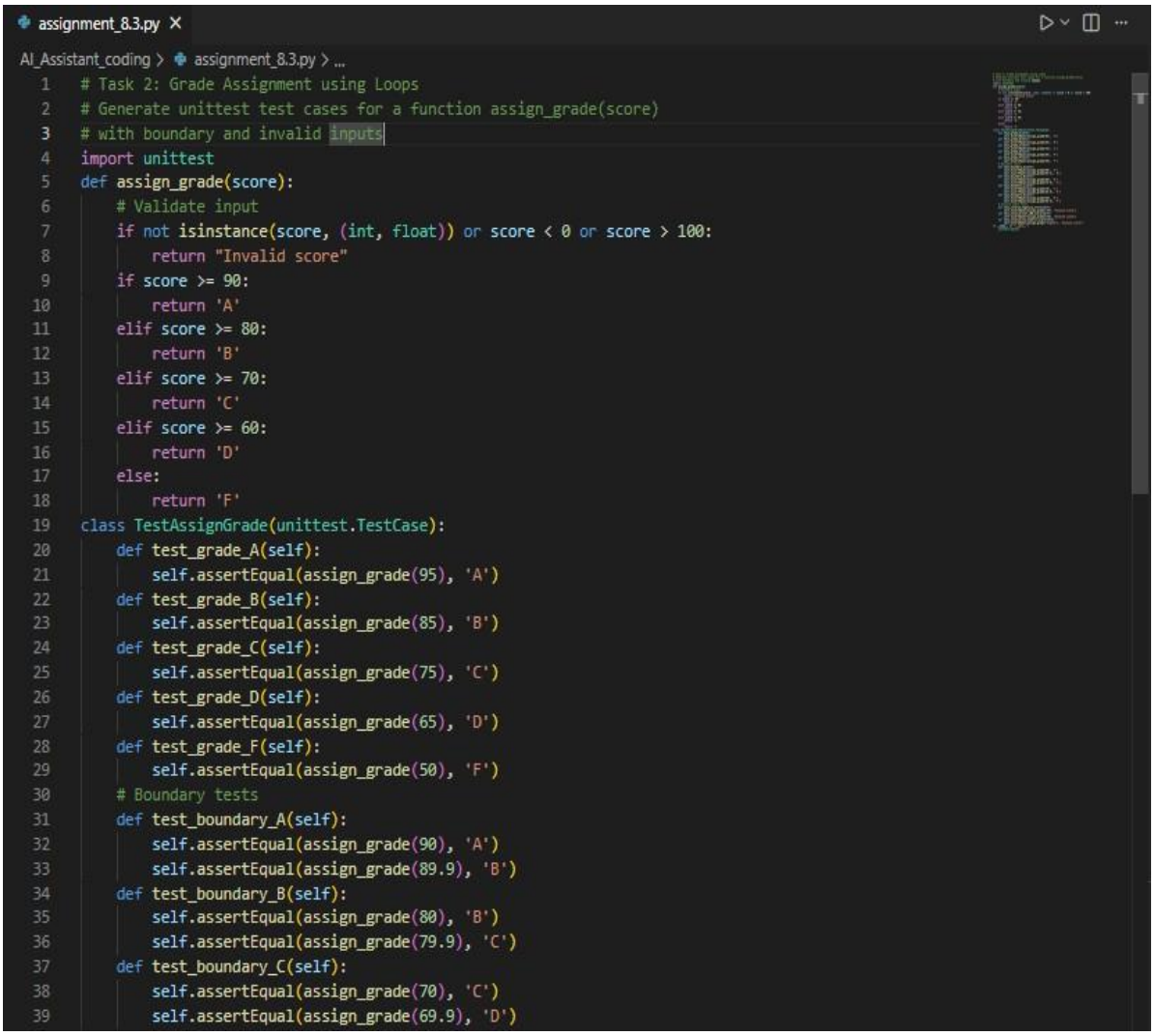
In this task, the AI began by creating test cases that included both valid and invalid email formats. The implementation was then developed to meet all the defined test conditions. The function verifies the presence of “@” and “.”, ensures that there is only one “@” symbol, and checks that the email does not begin or end with special characters. By following the Test-Driven Development (TDD) approach, the correctness of the solution was confirmed through successful test results.

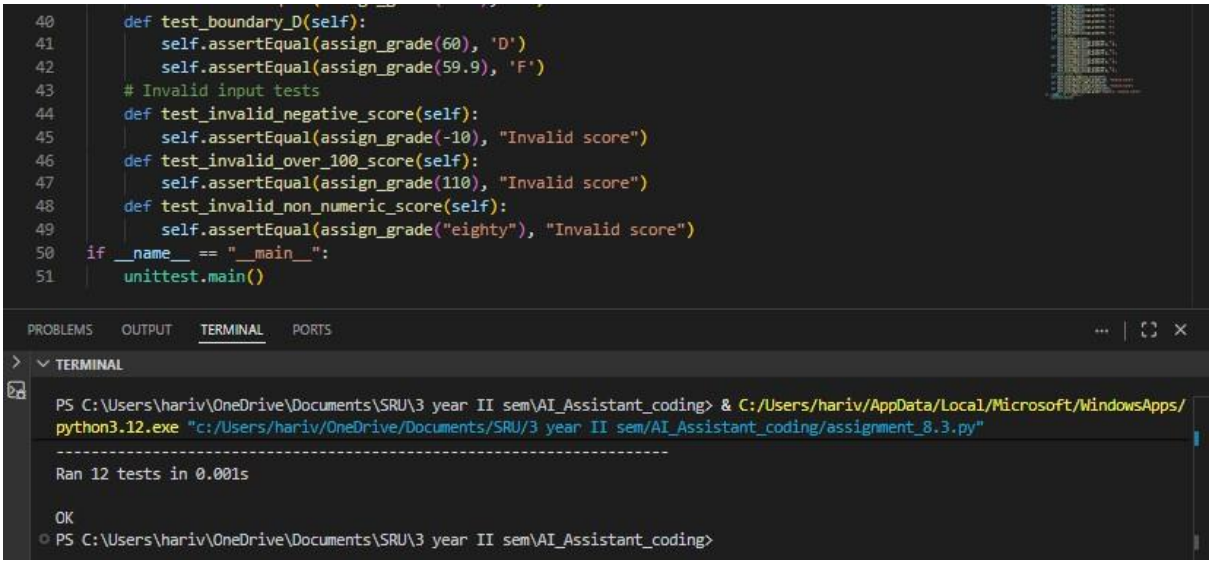
Task 2: Grade Assignment using Loops

Prompt:

Generate unit test test cases for a function assign_grade(score) with boundary and invalid inputs.

Code & Output:





Explanation:

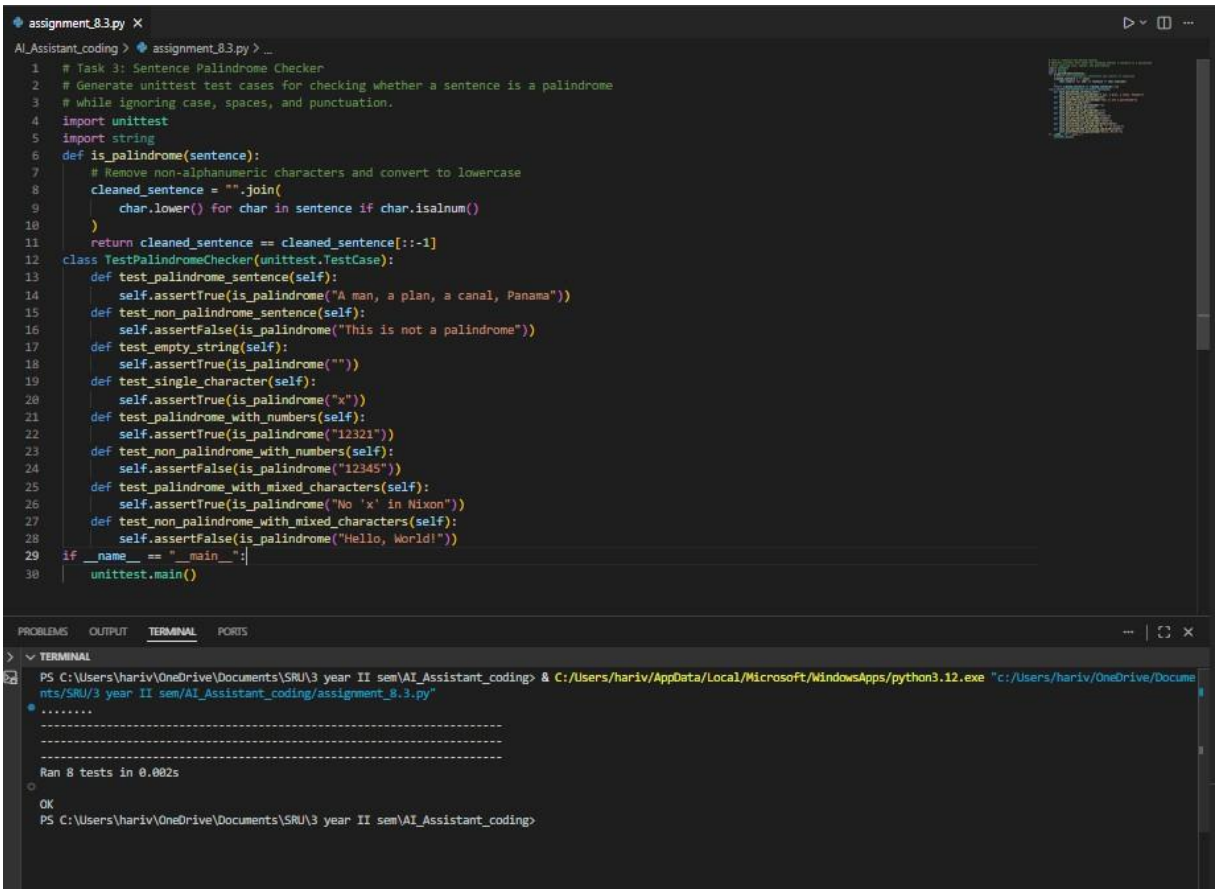
The AI-generated test cases covered normal score ranges, boundary conditions, and invalid inputs to ensure thorough evaluation. The implementation assigns grades accurately using conditional logic and also checks the input type and valid range to prevent incorrect or misleading results. Since all the test cases pass successfully, the solution's correctness is well confirmed.

Task 3: Sentence Palindrome Checker

Prompt:

Generate unittest test cases for checking whether a sentence is a palindrome while ignoring case, spaces, and punctuation.

Code & Output:



Explanation:

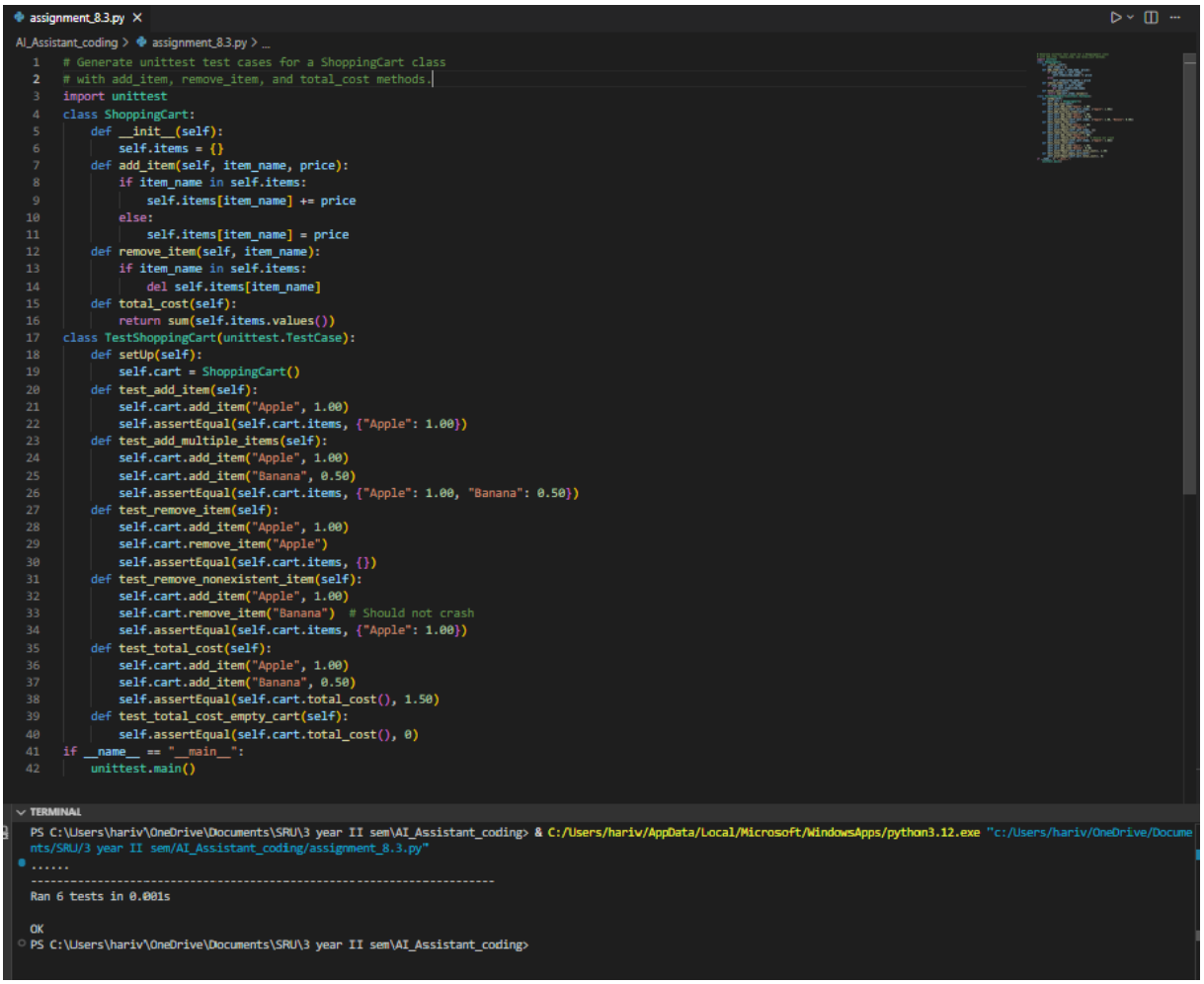
The AI-generated test cases cover both palindromic and non-palindromic sentences to ensure proper validation. In the implementation, spaces and punctuation are removed using regular expressions, and the text is converted to lowercase for consistency. The processed string is then compared with its reversed version. This approach guarantees accurate palindrome detection, regardless of the original formatting.

Task 4: ShoppingCart Class

Prompt:

Generate unittest test cases for a ShoppingCart class with add_item, remove_item, and total_cost methods.

Code & Output:



Explanation:

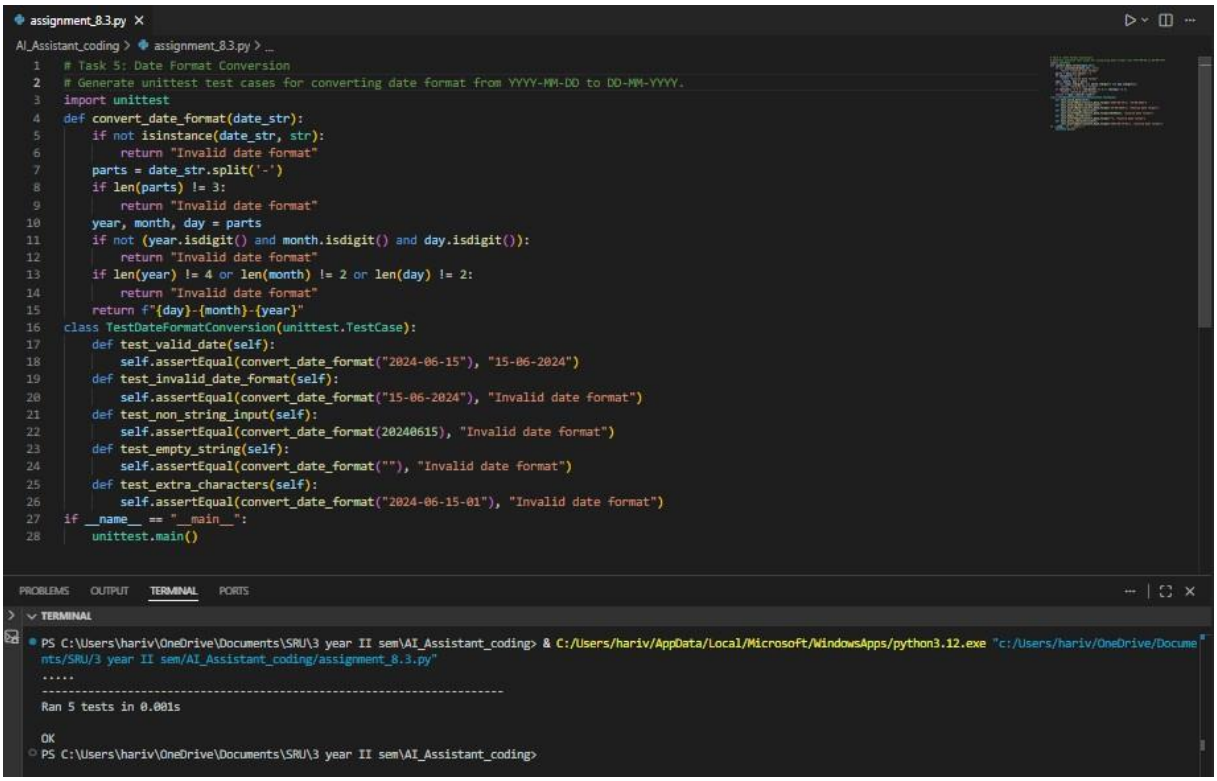
The AI-generated test cases thoroughly check item addition, item removal, and the calculation of the total cost. The class stores items and their prices in a dictionary, ensuring organized data management. Its methods properly update the cart's contents, and the total amount is calculated by summing all the stored values. Since the implementation successfully passes all the generated tests, its reliability is well established.

Task 5: Date Format Conversion

Prompt:

Generate unittest test cases for converting date format from YYYY-MM-DD to DD-MM-YYYY.

Code & Output:



```
# Task 5: Date Format Conversion
# Generate unittest test cases for converting date format from YYYY-MM-DD to DD-MM-YYYY.
import unittest
def convert_date_format(date_str):
    if not isinstance(date_str, str):
        return "Invalid date format"
    parts = date_str.split('-')
    if len(parts) != 3:
        return "Invalid date format"
    year, month, day = parts
    if not (year.isdigit() and month.isdigit() and day.isdigit()):
        return "Invalid date format"
    if len(year) != 4 or len(month) != 2 or len(day) != 2:
        return "Invalid date format"
    return f"{day}-{month}-{year}"
class TestDateFormatConversion(unittest.TestCase):
    def test_valid_date(self):
        self.assertEqual(convert_date_format("2024-06-15"), "15-06-2024")
    def test_invalid_date_format(self):
        self.assertEqual(convert_date_format("15-06-2024"), "Invalid date format")
    def test_non_string_input(self):
        self.assertEqual(convert_date_format(20240615), "Invalid date format")
    def test_empty_string(self):
        self.assertEqual(convert_date_format(""), "Invalid date format")
    def test_extra_characters(self):
        self.assertEqual(convert_date_format("2024-06-15-01"), "Invalid date format")
if __name__ == "__main__":
    unittest.main()
```

```
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding> & C:/Users/hariv/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/hariv/OneDrive/Documents/SRU/3 year II sem/AI_Assistant_coding/assignment_8.3.py"
.....
----------------------------------------------------------------------
Ran 5 tests in 0.001s

OK
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding>
```

Explanation:

The AI-generated test cases check both correctly formatted and invalid date inputs to ensure proper validation. The implementation separates the input string into parts and rearranges them into the desired format. It also includes error handling to return a clear message when the input format is incorrect. Since all the test cases pass successfully, the solution works as expected.

Final Conclusion:

This lab highlights the effectiveness of applying Test-Driven Development (TDD) with the support of AI. By creating test cases before writing the actual implementation, developers can better ensure correctness, reliability, and proper validation. While AI helps speed up the process of generating tests, human review is still crucial to guarantee comprehensive, meaningful, and well-designed test coverage.