Assignment-8.3

Test-Driven Development with AI – Generating and Working with TestCases

Name: V.Rithik Reddy

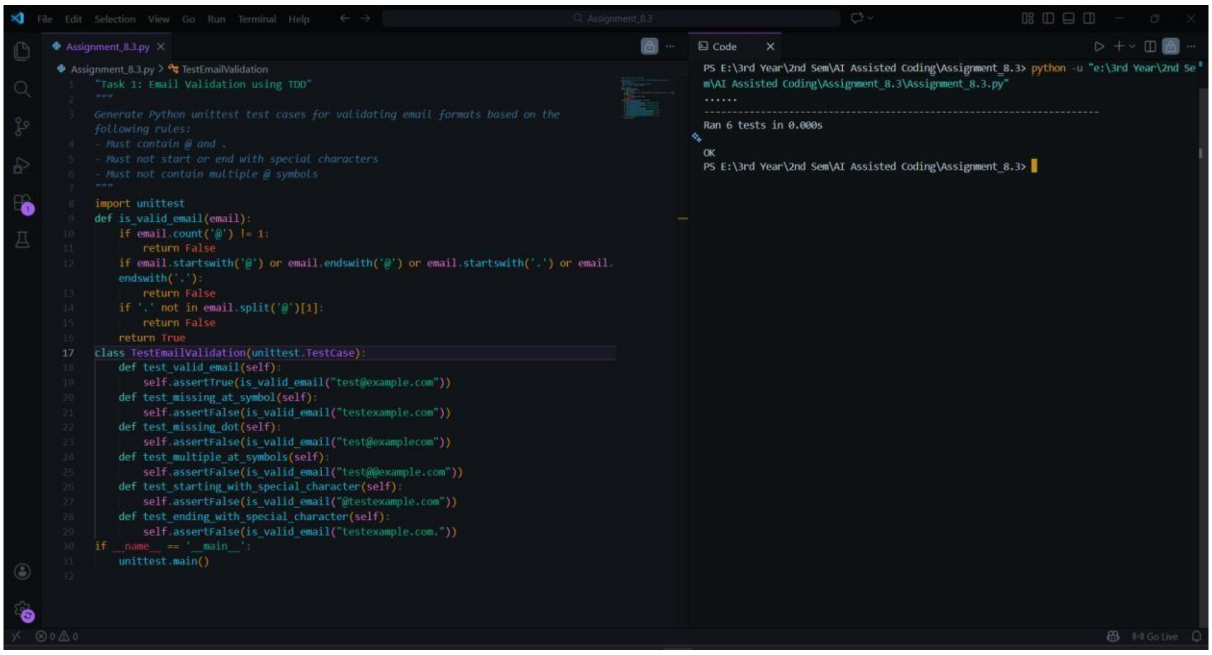
Hall ticket no: 2303A51263

Batch no: 19

Task 1: Email Validation using TDD 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:**



```
"Task 1: Email Validation using TDD"
"""
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
"""
import unittest
def is_valid_email(email):
    if email.count('@') != 1:
        return False
    if email.startswith('@') or email.endswith('@') or email.startswith('.') or email.
    endswith('.'):
        return False
    if '.' not in email.split('@')[1]:
        return False
    return True
class TestEmailValidation(unittest.TestCase):
    def test_valid_email(self):
        self.assertTrue(is_valid_email("test@example.com"))
    def test_missing_at_symbol(self):
        self.assertFalse(is_valid_email("testexample.com"))
    def test_missing_dot(self):
        self.assertFalse(is_valid_email("test@examplecom"))
    def test_multiple_at_symbols(self):
        self.assertFalse(is_valid_email("test@@example.com"))
    def test_starting_with_special_character(self):
        self.assertFalse(is_valid_email("@testexample.com"))
    def test_ending_with_special_character(self):
        self.assertFalse(is_valid_email("testexample.com."))
if __name__ == '__main__':
    unittest.main()
```

```
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_8.3> python -u "e:\3rd Year\2nd Se
m\AI Assisted Coding\Assignment_8.3\Assignment_8.3.py"
......
----------------------------------------------------------------------
Ran 6 tests in 0.000s

OK
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_8.3>
```

Explanation:

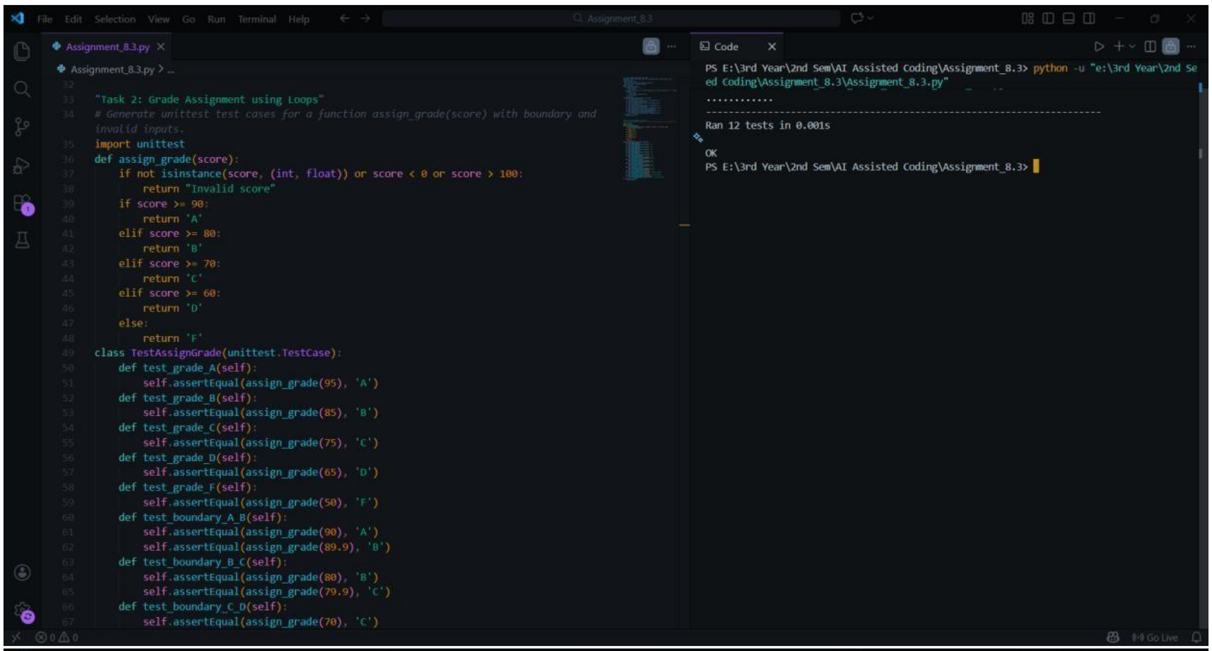
In this task, AI first generated test cases covering both valid and invalid email formats. The implementation was then written to satisfy all test conditions. The function checks for the presence of “@” and “.”, prevents multiple “@” symbols, and ensures the email does not start or end with special characters. By following the TDD approach, correctness was validated through successful test execution.

Task 2: Grade Assignment using Loops

Prompt:

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

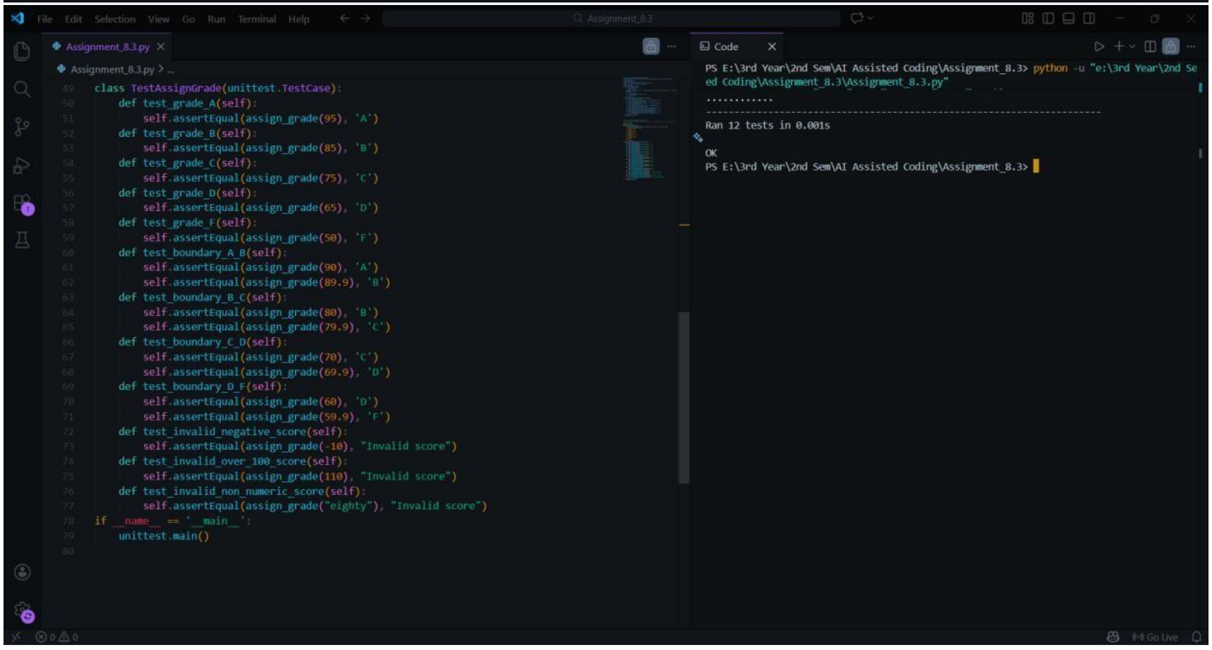
Code & Output:



```python
"Task 2: Grade Assignment using Loops"
# Generate unittest test cases for a function assign_grade(score) with boundary and invalid inputs.
import unittest
def assign_grade(score):
    if not isinstance(score, (int, float)) or score < 0 or score > 100:
        return "Invalid score"
    if score >= 90:
        return 'A'
    elif score >= 80:
        return 'B'
    elif score >= 70:
        return 'C'
    elif score >= 60:
        return 'D'
    else:
        return 'F'
class TestAssignGrade(unittest.TestCase):
    def test_grade_A(self):
        self.assertEqual(assign_grade(95), 'A')
    def test_grade_B(self):
        self.assertEqual(assign_grade(85), 'B')
    def test_grade_C(self):
        self.assertEqual(assign_grade(75), 'C')
    def test_grade_D(self):
        self.assertEqual(assign_grade(65), 'D')
    def test_grade_F(self):
        self.assertEqual(assign_grade(50), 'F')
    def test_boundary_A_B(self):
        self.assertEqual(assign_grade(90), 'A')
        self.assertEqual(assign_grade(89.9), 'B')
    def test_boundary_B_C(self):
        self.assertEqual(assign_grade(80), 'B')
        self.assertEqual(assign_grade(79.9), 'C')
    def test_boundary_C_D(self):
        self.assertEqual(assign_grade(70), 'C')
```

```
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_8.3> python -u "e:\3rd Year\2nd Se
ed Coding\Assignment_8.3\Assignment_8.3.py"
............
----------------------------------------------------------------------
Ran 12 tests in 0.001s

OK
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_8.3>
```



```python
class TestAssignGrade(unittest.TestCase):
    def test_grade_A(self):
        self.assertEqual(assign_grade(95), 'A')
    def test_grade_B(self):
        self.assertEqual(assign_grade(85), 'B')
    def test_grade_C(self):
        self.assertEqual(assign_grade(75), 'C')
    def test_grade_D(self):
        self.assertEqual(assign_grade(65), 'D')
    def test_grade_F(self):
        self.assertEqual(assign_grade(50), 'F')
    def test_boundary_A_B(self):
        self.assertEqual(assign_grade(90), 'A')
        self.assertEqual(assign_grade(89.9), 'B')
    def test_boundary_B_C(self):
        self.assertEqual(assign_grade(80), 'B')
        self.assertEqual(assign_grade(79.9), 'C')
    def test_boundary_C_D(self):
        self.assertEqual(assign_grade(70), 'C')
        self.assertEqual(assign_grade(69.9), 'D')
    def test_boundary_D_F(self):
        self.assertEqual(assign_grade(60), 'D')
        self.assertEqual(assign_grade(59.9), 'F')
    def test_invalid_negative_score(self):
        self.assertEqual(assign_grade(-10), "Invalid score")
    def test_invalid_over_100_score(self):
        self.assertEqual(assign_grade(110), "Invalid score")
    def test_invalid_non_numeric_score(self):
        self.assertEqual(assign_grade("eighty"), "Invalid score")
if __name__ == '__main__':
    unittest.main()
```

```
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_8.3> python -u "e:\3rd Year\2nd Se
ed Coding\Assignment_8.3\Assignment_8.3.py"
............
----------------------------------------------------------------------
Ran 12 tests in 0.001s

OK
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_8.3>
```

Explanation:

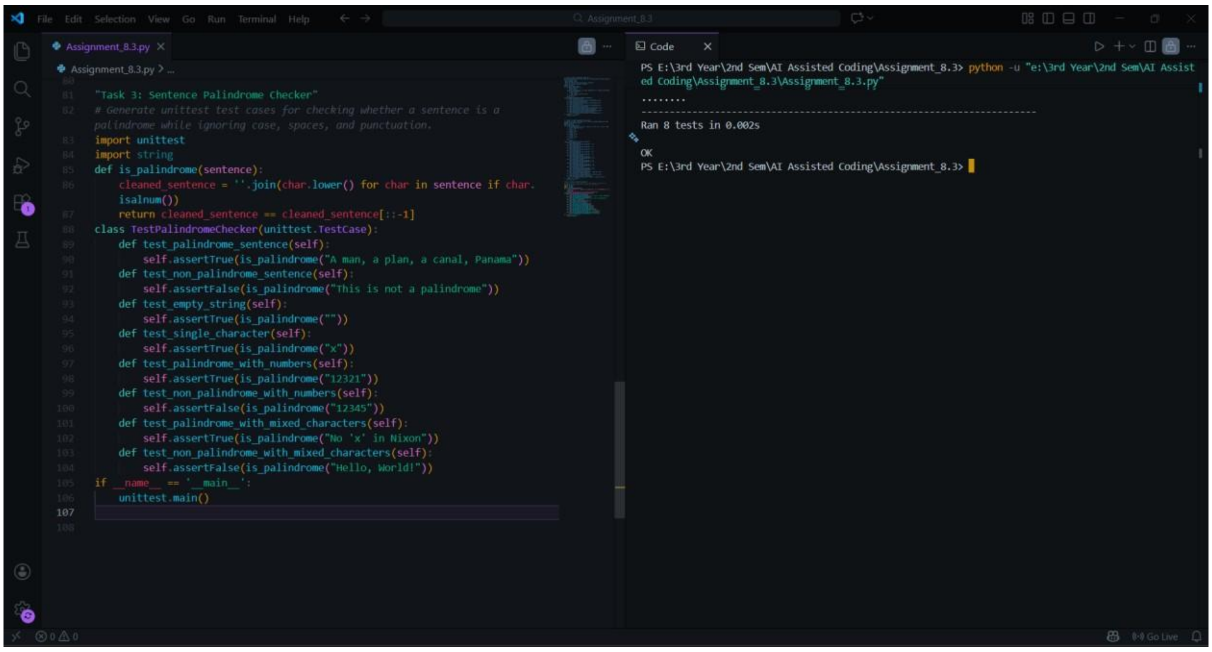
AI-generated tests included normal ranges, boundary values, and invalid inputs. The implementation ensures proper grade assignment using conditional logic. It also validates input type and range, preventing incorrect values from producing misleading grades. All test cases pass successfully, confirming correctness.

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:



```python
"Task 3: Sentence Palindrome Checker"
# Generate unittest test cases for checking whether a sentence is a palindrome while ignoring case, spaces, and punctuation.
import unittest
import string
def is_palindrome(sentence):
    cleaned_sentence = ''.join(char.lower() for char in sentence if char.isalnum())
    return cleaned_sentence == cleaned_sentence[::-1]
class TestPalindromeChecker(unittest.TestCase):
    def test_palindrome_sentence(self):
        self.assertTrue(is_palindrome("A man, a plan, a canal, Panama"))
    def test_non_palindrome_sentence(self):
        self.assertFalse(is_palindrome("This is not a palindrome"))
    def test_empty_string(self):
        self.assertTrue(is_palindrome(""))
    def test_single_character(self):
        self.assertTrue(is_palindrome("x"))
    def test_palindrome_with_numbers(self):
        self.assertTrue(is_palindrome("12321"))
    def test_non_palindrome_with_numbers(self):
        self.assertFalse(is_palindrome("12345"))
    def test_palindrome_with_mixed_characters(self):
        self.assertTrue(is_palindrome("No 'x' in Nixon"))
    def test_non_palindrome_with_mixed_characters(self):
        self.assertFalse(is_palindrome("Hello, World!"))
if __name__ == '__main__':
    unittest.main()
```

```
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_8.3> python -u "e:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_8.3\Assignment_8.3.py"
........
----------------------------------------------------------------------
Ran 8 tests in 0.002s

OK
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_8.3>
```

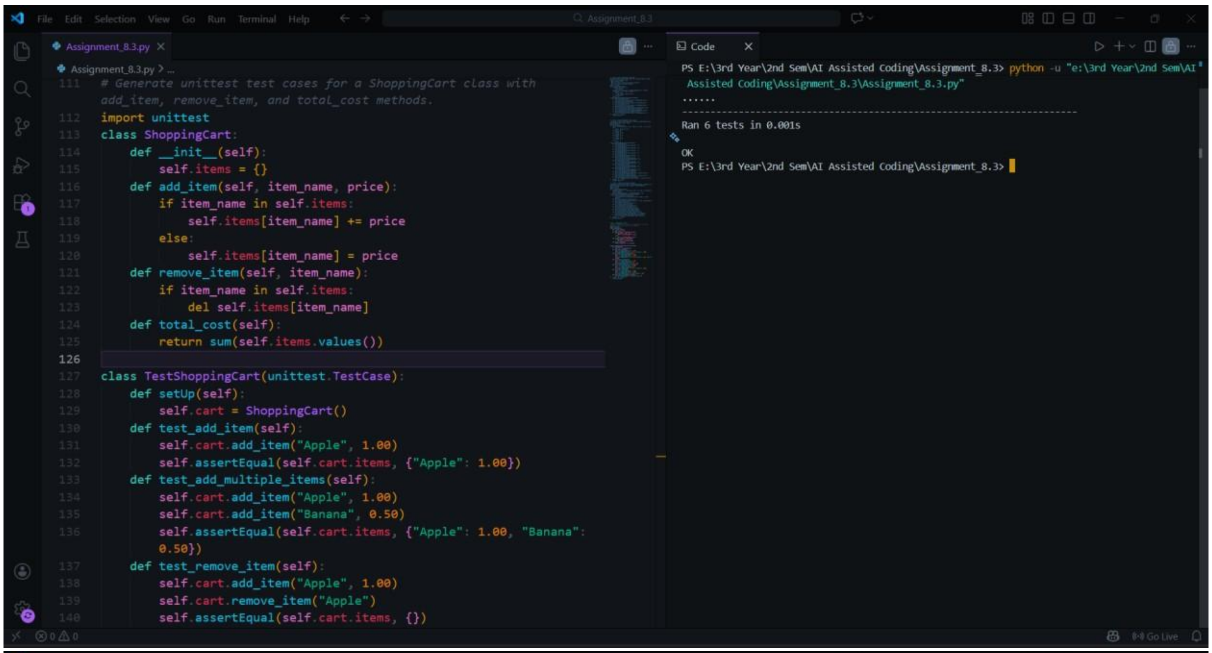
Explanation:

The AI-generated tests check both palindromic and non-palindromic sentences. The implementation removes spaces and punctuation using regular expressions and converts text to lowercase. The cleaned string is compared with its reverse. This ensures accurate palindrome detection regardless of 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:

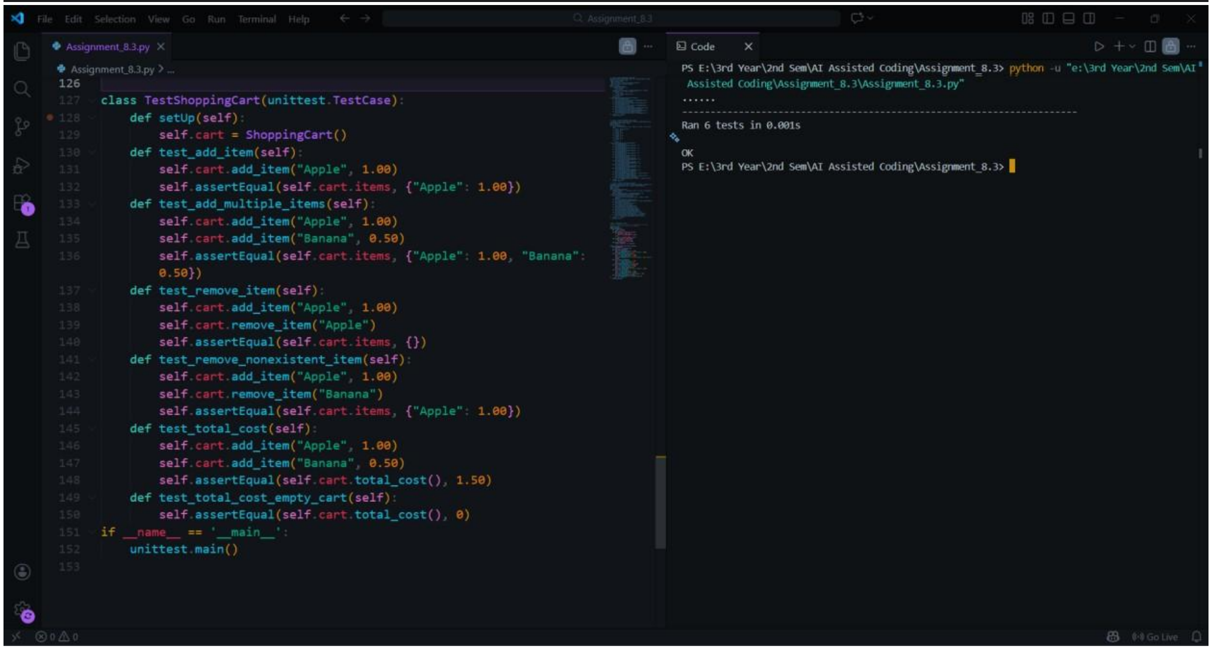


```python
# Generate unittest test cases for a ShoppingCart class with
add_item, remove_item, and total_cost methods.
import unittest
class ShoppingCart:
    def __init__(self):
        self.items = {}
    def add_item(self, item_name, price):
        if item_name in self.items:
            self.items[item_name] += price
        else:
            self.items[item_name] = price
    def remove_item(self, item_name):
        if item_name in self.items:
            del self.items[item_name]
    def total_cost(self):
        return sum(self.items.values())

class TestShoppingCart(unittest.TestCase):
    def setUp(self):
        self.cart = ShoppingCart()
    def test_add_item(self):
        self.cart.add_item("Apple", 1.00)
        self.assertEqual(self.cart.items, {"Apple": 1.00})
    def test_add_multiple_items(self):
        self.cart.add_item("Apple", 1.00)
        self.cart.add_item("Banana", 0.50)
        self.assertEqual(self.cart.items, {"Apple": 1.00, "Banana":
        0.50})
    def test_remove_item(self):
        self.cart.add_item("Apple", 1.00)
        self.cart.remove_item("Apple")
        self.assertEqual(self.cart.items, {})
```

```
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_8.3> python -u "e:\3rd Year\2nd Sem\AI
 Assisted Coding\Assignment_8.3\Assignment_8.3.py"
......
----------------------------------------------------------------------
Ran 6 tests in 0.001s

OK
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_8.3>
```



```python

class TestShoppingCart(unittest.TestCase):
    def setUp(self):
        self.cart = ShoppingCart()
    def test_add_item(self):
        self.cart.add_item("Apple", 1.00)
        self.assertEqual(self.cart.items, {"Apple": 1.00})
    def test_add_multiple_items(self):
        self.cart.add_item("Apple", 1.00)
        self.cart.add_item("Banana", 0.50)
        self.assertEqual(self.cart.items, {"Apple": 1.00, "Banana":
        0.50})
    def test_remove_item(self):
        self.cart.add_item("Apple", 1.00)
        self.cart.remove_item("Apple")
        self.assertEqual(self.cart.items, {})
    def test_remove_nonexistent_item(self):
        self.cart.add_item("Apple", 1.00)
        self.cart.remove_item("Banana")
        self.assertEqual(self.cart.items, {"Apple": 1.00})
    def test_total_cost(self):
        self.cart.add_item("Apple", 1.00)
        self.cart.add_item("Banana", 0.50)
        self.assertEqual(self.cart.total_cost(), 1.50)
    def test_total_cost_empty_cart(self):
        self.assertEqual(self.cart.total_cost(), 0)
if __name__ == '__main__':
    unittest.main()
```

```
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_8.3> python -u "e:\3rd Year\2nd Sem\AI
 Assisted Coding\Assignment_8.3\Assignment_8.3.py"
......
----------------------------------------------------------------------
Ran 6 tests in 0.001s

OK
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_8.3>
```

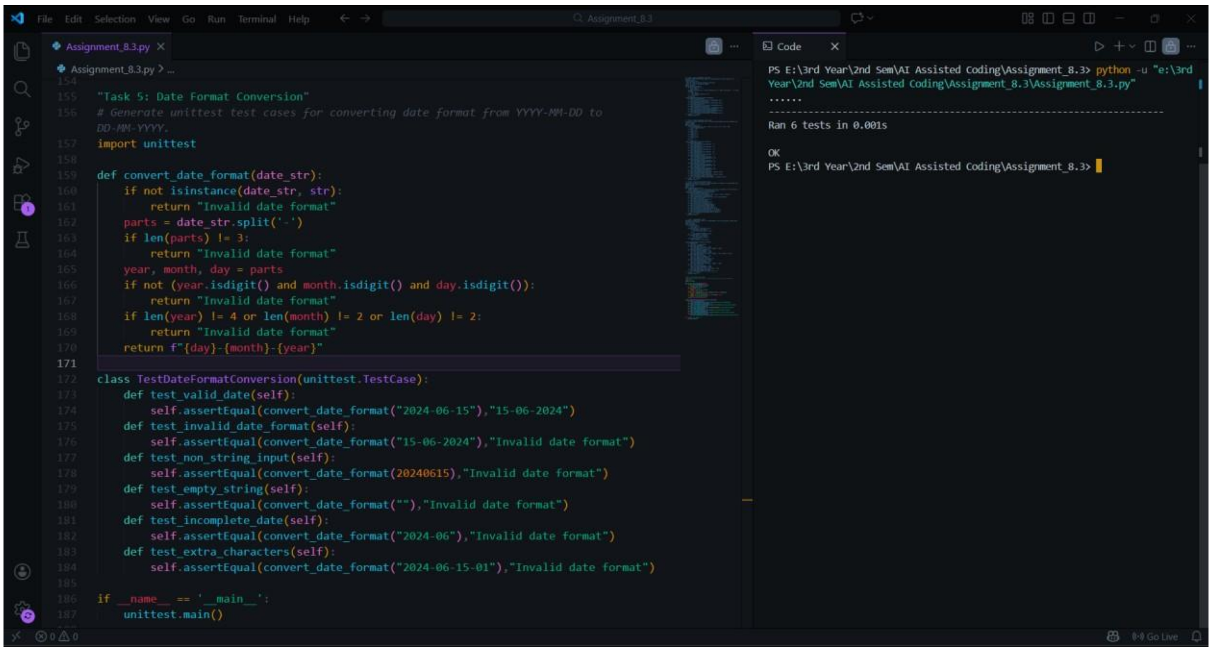
Explanation:

The AI-generated test cases validate item addition, removal, and total cost calculation. The class uses a dictionary to store items and prices. Methods correctly update the cart state, and the total cost is computed using the sum of values. The implementation passes all generated tests, confirming reliability.

Task 5: Date Format Conversion Prompt:

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

Code & Output:



```python
"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_incomplete_date(self):
        self.assertEqual(convert_date_format("2024-06"),"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 E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_8.3> python -u "e:\3rd
Year\2nd Sem\AI Assisted Coding\Assignment_8.3\Assignment_8.3.py"
......
----------------------------------------------------------------------
Ran 6 tests in 0.001s

OK
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_8.3>
```

Explanation:

The AI-generated tests verify both valid and invalid date inputs. The implementation splits the input string and rearranges the components into the required format. Error handling ensures that incorrectly formatted inputs return an appropriate message. All test cases pass successfully.

Final Conclusion:

This lab demonstrates the effectiveness of Test-Driven Development using AI. By generating test cases first, developers can ensure correctness, reliability, and validation before implementation. AI accelerates test creation, but human review remains essential for designing robust and meaningful test coverage.