| | |
|---|---|
| **SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE** | **DEPARTMENT OF COMPUTER SCIENCE ENGINEERING** |

| | | |
|---|---|---|
| **ProgramName:**B. Tech | **Assignment Type: Lab** | **AcademicYear:**2025-2026 |

| | | | |
|---|---|---|---|
| **CourseCoordinatorName** | Venkataramana Veeramsetty | | |
| **Instructor(s)Name** | Dr. V. Venkataramana (Co-ordinator)<br>Dr. T. Sampath Kumar<br>Dr. Pramoda Patro<br>Dr. Brij Kishor Tiwari<br>Dr.J.Ravichander<br>Dr. Mohammand Ali Shaik<br>Dr. Anirodh Kumar<br>Mr. S.Naresh Kumar<br>Dr. RAJESH VELPULA<br>Mr. Kundhan Kumar<br>Ms. Ch.Rajitha<br>Mr. M Prakash<br>Mr. B.Raju<br>Intern 1 (Dharma teja)<br>Intern 2 (Sai Prasad)<br>Intern 3 (Sowmya)<br>NS_2 ( Mounika) | | |
| **CourseCode** | 24CS002PC215 | **CourseTitle** | AI Assisted Coding |
| **Year/Sem** | II/I | **Regulation** | R24 |
| **Date and Day of Assignment** | Week4 - Wednesday | **Time(s)** | |
| **Duration** | 2 Hours | **Applicableto Batches** | |
| **AssignmentNumber:8.3**(Present assignment number)/**24**(Total number of assignments) | | | |

| **Q.No.** | **Question** | ***ExpectedTime to complete*** |
|---|---|---|
| 1 | Lab 8: Test-Driven Development with AI – Generating and Working with Test Cases<br><br>**Lab Objectives:**<br>• To introduce students to test-driven development (TDD) using AI code generation tools.<br>• To enable the generation of test cases before writing code implementations. | Week4 - Wednesday |

- To reinforce the importance of testing, validation, and error handling.
- To encourage writing clean and reliable code based on AI-generated test expectations.

**Lab Outcomes (LOs):**
After completing this lab, students will be able to:

- Use AI tools to write test cases for Python functions and classes.
- Implement functions based on test cases in a test-first development style.
- Use unittest or pytest to validate code correctness.
- Analyze the completeness and coverage of AI-generated tests.
- Compare AI-generated and manually written test cases for quality and logic

**Task Description#1**

Use AI to generate test cases for is_valid_email(email) and then implement the validator function.

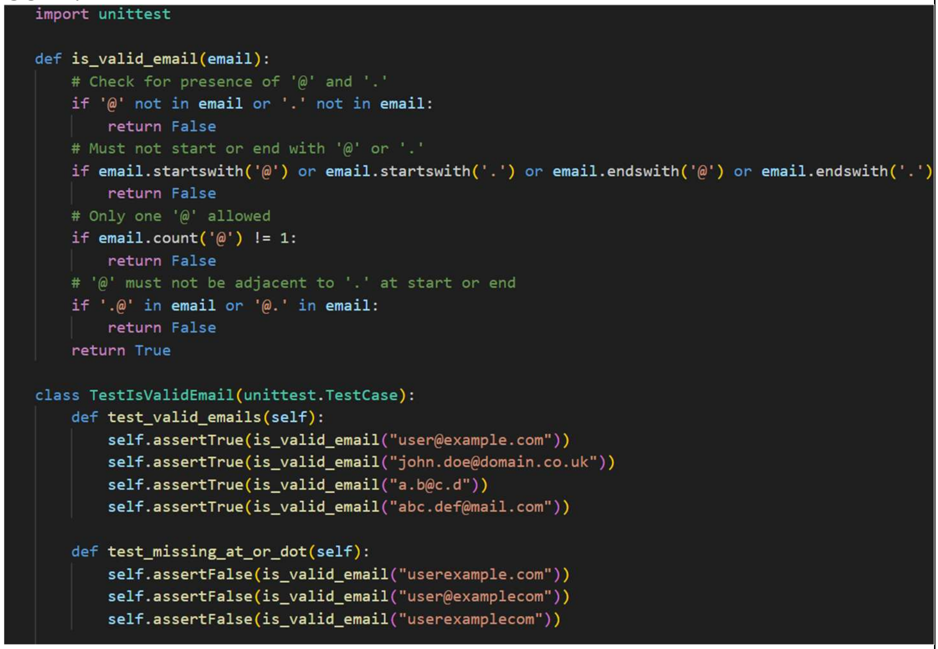
**Requirements:**

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

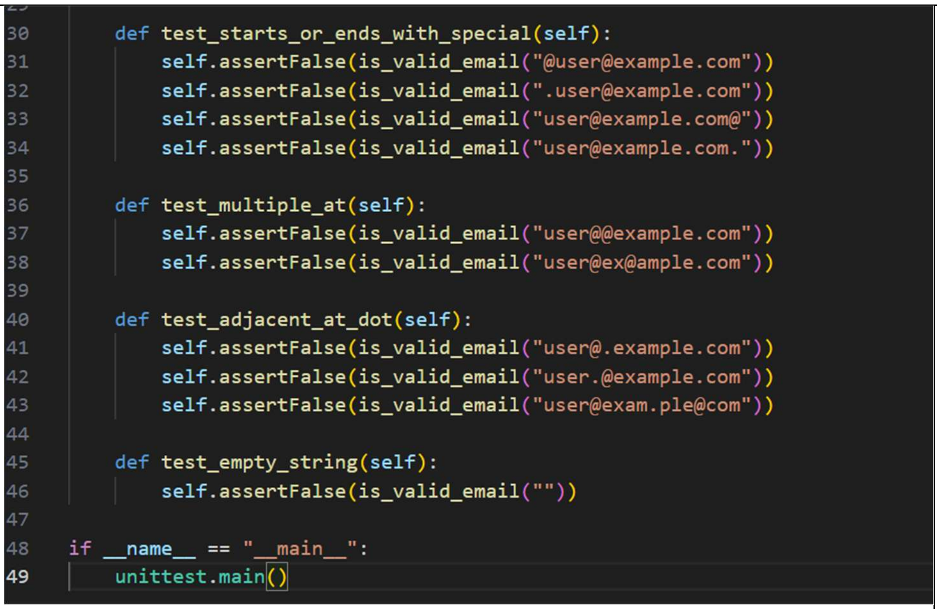
**Expected Output#1**

- Email validation logic passing all test cases

**CODE:**

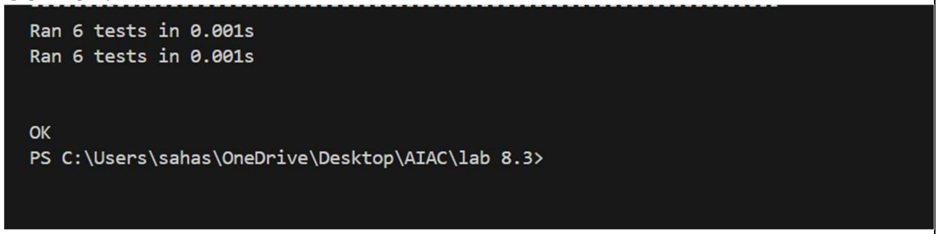


```python
import unittest

def is_valid_email(email):
    # Check for presence of '@' and '.'
    if '@' not in email or '.' not in email:
        return False
    # Must not start or end with '@' or '.'
    if email.startswith('@') or email.startswith('.') or email.endswith('@') or email.endswith('.')
        return False
    # Only one '@' allowed
    if email.count('@') != 1:
        return False
    # '@' must not be adjacent to '.' at start or end
    if '.@' in email or '@.' in email:
        return False
    return True

class TestIsValidEmail(unittest.TestCase):
    def test_valid_emails(self):
        self.assertTrue(is_valid_email("user@example.com"))
        self.assertTrue(is_valid_email("john.doe@domain.co.uk"))
        self.assertTrue(is_valid_email("a.b@c.d"))
        self.assertTrue(is_valid_email("abc.def@mail.com"))

    def test_missing_at_or_dot(self):
        self.assertFalse(is_valid_email("userexample.com"))
        self.assertFalse(is_valid_email("user@examplecom"))
        self.assertFalse(is_valid_email("userexamplecom"))
```



```python
    def test_starts_or_ends_with_special(self):
        self.assertFalse(is_valid_email("@user@example.com"))
        self.assertFalse(is_valid_email(".user@example.com"))
        self.assertFalse(is_valid_email("user@example.com@"))
        self.assertFalse(is_valid_email("user@example.com."))

    def test_multiple_at(self):
        self.assertFalse(is_valid_email("user@@example.com"))
        self.assertFalse(is_valid_email("user@ex@ample.com"))

    def test_adjacent_at_dot(self):
        self.assertFalse(is_valid_email("user@.example.com"))
        self.assertFalse(is_valid_email("user.@example.com"))
        self.assertFalse(is_valid_email("user@exam.ple@com"))

    def test_empty_string(self):
        self.assertFalse(is_valid_email(""))

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

**OUTPUT:**



```
Ran 6 tests in 0.001s
Ran 6 tests in 0.001s


OK
PS C:\Users\sahas\OneDrive\Desktop\AIAC\lab 8.3>
```

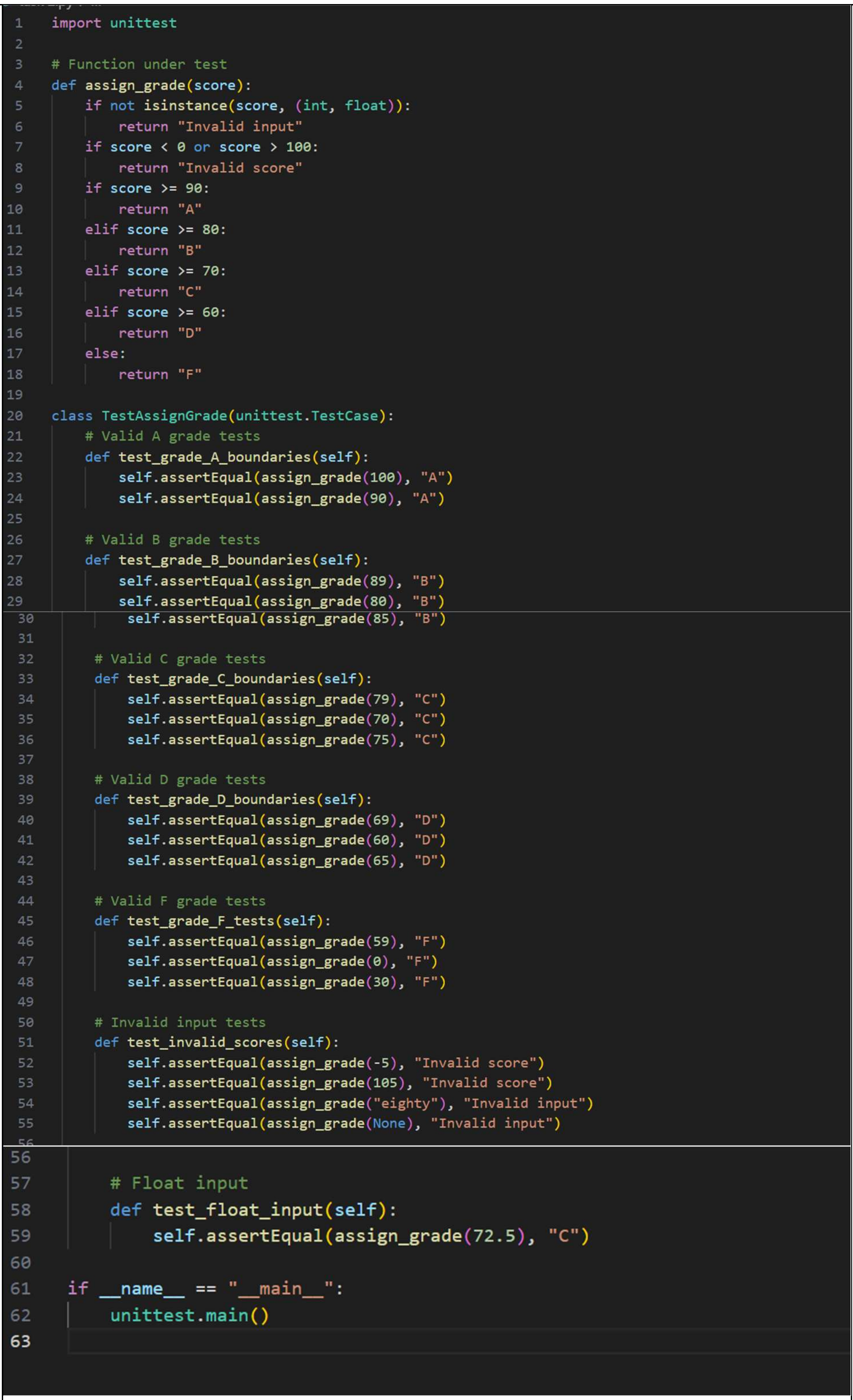
**Task Description#2 (Loops)**

- Ask AI to generate test cases for assign_grade(score) function. Handle boundary and invalid inputs.
  **Requirements**
- AI should generate test cases for assign_grade(score) where: 90-100: A, 80-89: B, 70-79: C, 60-69: D, <60: F
- Include boundary values and invalid inputs (e.g., -5, 105, "eighty").

**Expected Output#2**

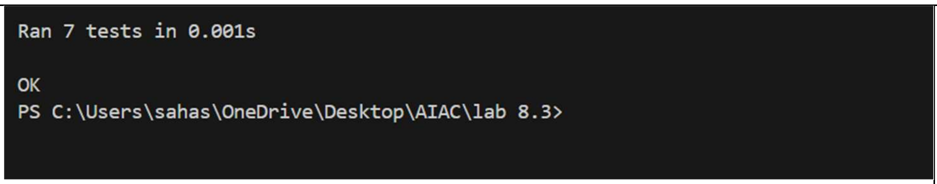
Grade assignment function passing test suite

CODE:



```python
import unittest

# Function under test
def assign_grade(score):
    if not isinstance(score, (int, float)):
        return "Invalid input"
    if 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):
    # Valid A grade tests
    def test_grade_A_boundaries(self):
        self.assertEqual(assign_grade(100), "A")
        self.assertEqual(assign_grade(90), "A")

    # Valid B grade tests
    def test_grade_B_boundaries(self):
        self.assertEqual(assign_grade(89), "B")
        self.assertEqual(assign_grade(80), "B")
        self.assertEqual(assign_grade(85), "B")

    # Valid C grade tests
    def test_grade_C_boundaries(self):
        self.assertEqual(assign_grade(79), "C")
        self.assertEqual(assign_grade(70), "C")
        self.assertEqual(assign_grade(75), "C")

    # Valid D grade tests
    def test_grade_D_boundaries(self):
        self.assertEqual(assign_grade(69), "D")
        self.assertEqual(assign_grade(60), "D")
        self.assertEqual(assign_grade(65), "D")

    # Valid F grade tests
    def test_grade_F_tests(self):
        self.assertEqual(assign_grade(59), "F")
        self.assertEqual(assign_grade(0), "F")
        self.assertEqual(assign_grade(30), "F")

    # Invalid input tests
    def test_invalid_scores(self):
        self.assertEqual(assign_grade(-5), "Invalid score")
        self.assertEqual(assign_grade(105), "Invalid score")
        self.assertEqual(assign_grade("eighty"), "Invalid input")
        self.assertEqual(assign_grade(None), "Invalid input")

    # Float input
    def test_float_input(self):
        self.assertEqual(assign_grade(72.5), "C")

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

```

**OUTPUT:**



```
Ran 7 tests in 0.001s

OK
PS C:\Users\sahas\OneDrive\Desktop\AIAC\lab 8.3>
```

**Task Description#3**

- Generate test cases using AI for is_sentence_palindrome(sentence). Ignore case, punctuation, and spaces

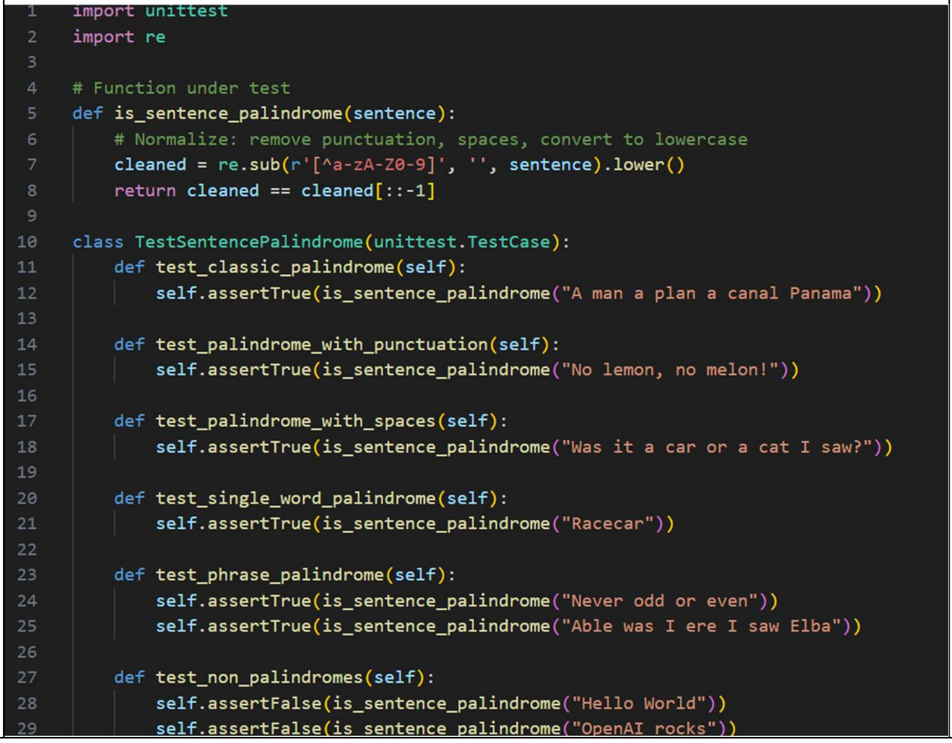
  **Requirement**
- Ask AI to create test cases for is_sentence_palindrome(sentence) (ignores case, spaces, and punctuation).

- Example:
  "A man a plan a canal Panama" → True

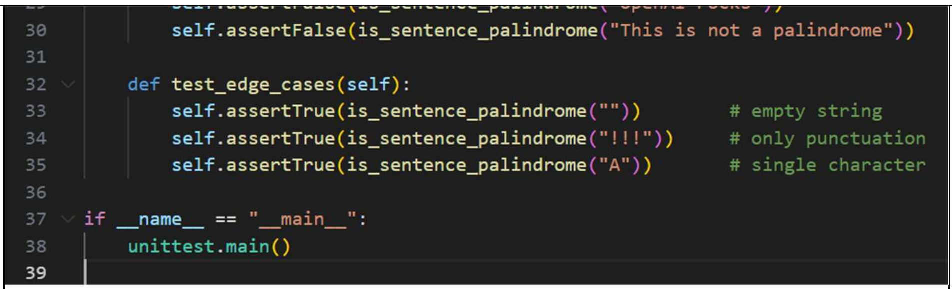
**Expected Output#3**

- Function returns True/False for cleaned sentences
- Implement the function to pass AI-generated tests

CODE:

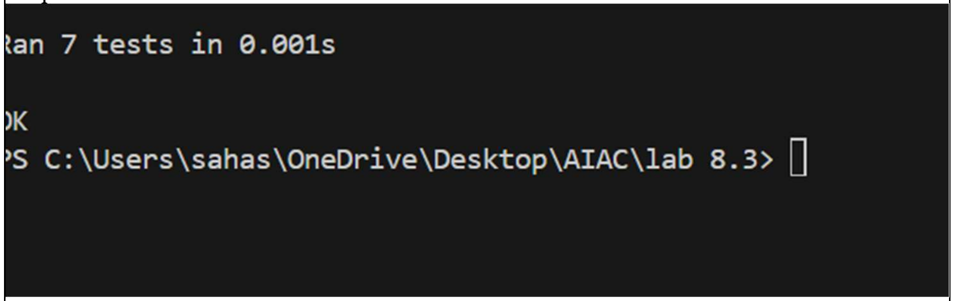


```python
import unittest
import re

# Function under test
def is_sentence_palindrome(sentence):
    # Normalize: remove punctuation, spaces, convert to lowercase
    cleaned = re.sub(r'[^a-zA-Z0-9]', '', sentence).lower()
    return cleaned == cleaned[::-1]

class TestSentencePalindrome(unittest.TestCase):
    def test_classic_palindrome(self):
        self.assertTrue(is_sentence_palindrome("A man a plan a canal Panama"))

    def test_palindrome_with_punctuation(self):
        self.assertTrue(is_sentence_palindrome("No lemon, no melon!"))

    def test_palindrome_with_spaces(self):
        self.assertTrue(is_sentence_palindrome("Was it a car or a cat I saw?"))

    def test_single_word_palindrome(self):
        self.assertTrue(is_sentence_palindrome("Racecar"))

    def test_phrase_palindrome(self):
        self.assertTrue(is_sentence_palindrome("Never odd or even"))
        self.assertTrue(is_sentence_palindrome("Able was I ere I saw Elba"))

    def test_non_palindromes(self):
        self.assertFalse(is_sentence_palindrome("Hello World"))
        self.assertFalse(is_sentence_palindrome("OpenAI rocks"))
```



```
        self.assertFalse(is_sentence_palindrome("This is not a palindrome"))

    def test_edge_cases(self):
        self.assertTrue(is_sentence_palindrome(""))        # empty string
        self.assertTrue(is_sentence_palindrome("!!!"))     # only punctuation
        self.assertTrue(is_sentence_palindrome("A"))       # single character

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

Output:



```
Ran 7 tests in 0.001s

OK
PS C:\Users\sahas\OneDrive\Desktop\AIAC\lab 8.3>
```

**Task Description#4**

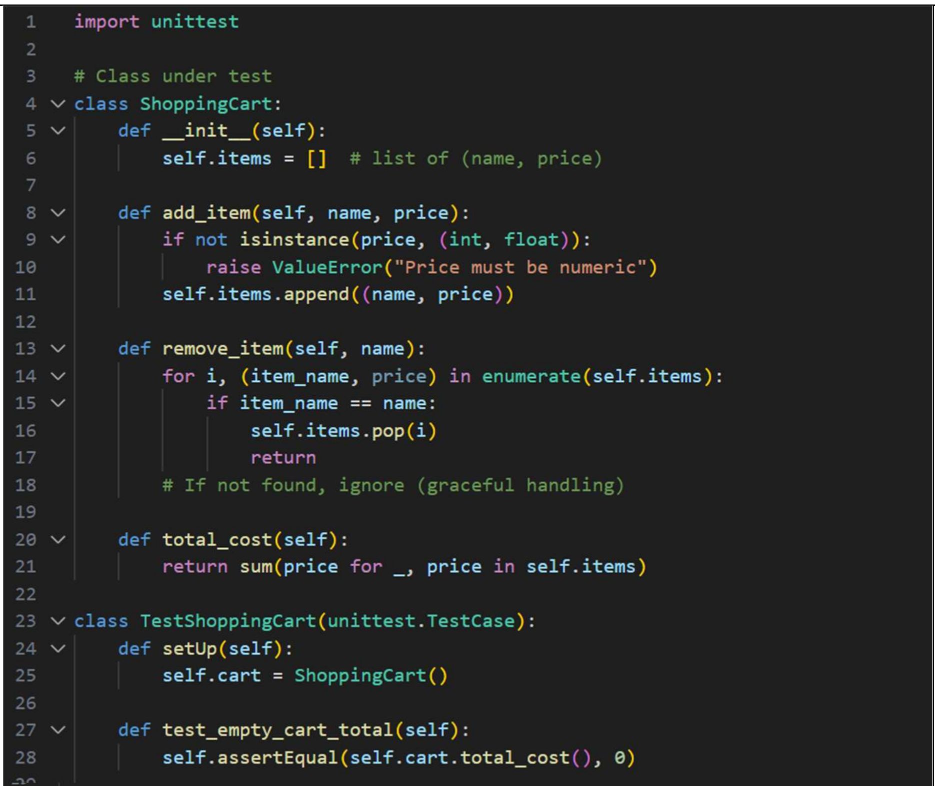
- Let AI fix it Prompt AI to generate test cases for a ShoppingCart class (add_item, remove_item, total_cost).

  **Methods:**
  Add_item(name,orice)
  Remove_item(name)
  Total_cost()

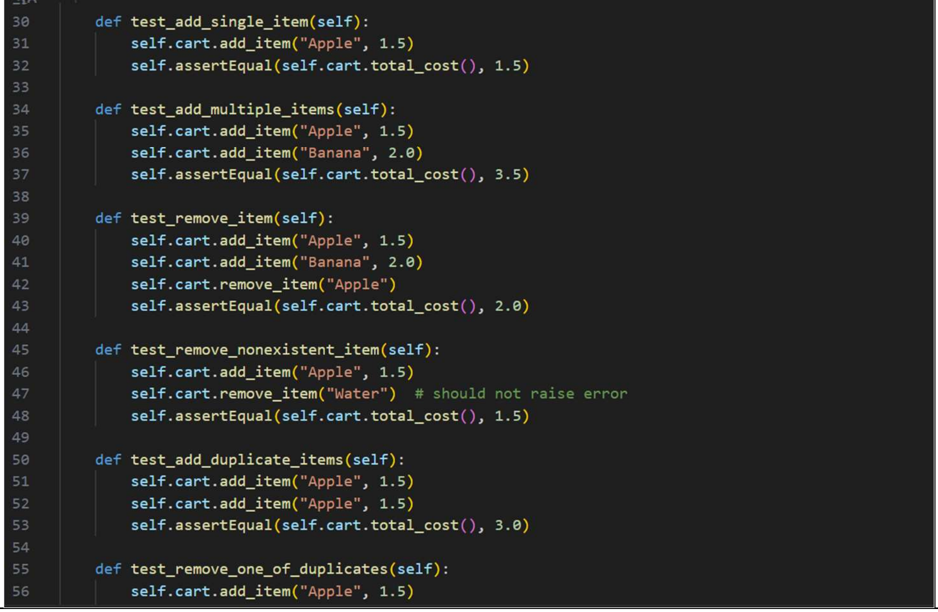
**Expected Output#4**

- Full class with tested functionalities

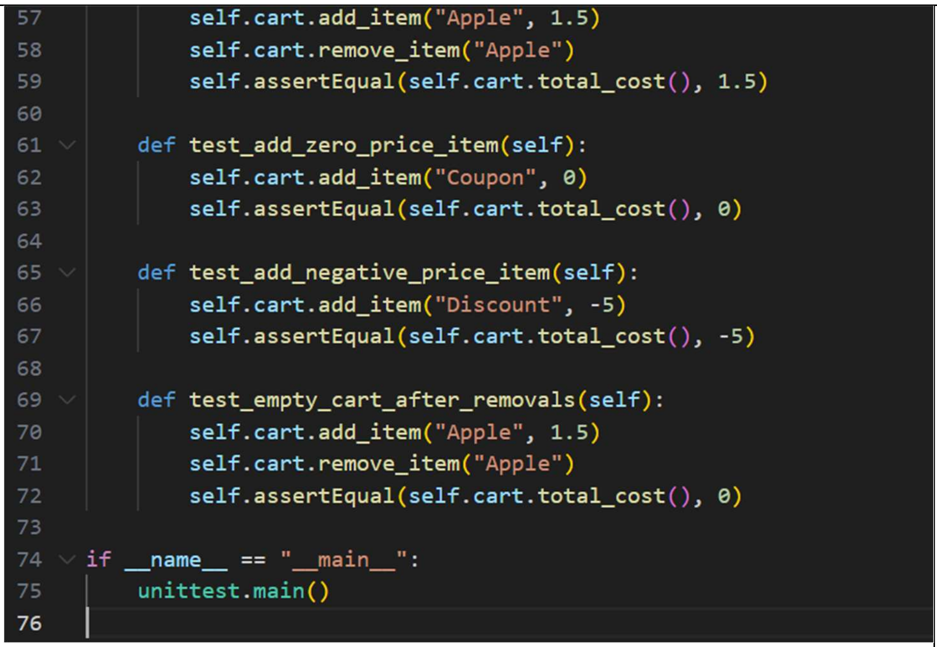
CODE:



```python
import unittest

# Class under test
class ShoppingCart:
    def __init__(self):
        self.items = []  # list of (name, price)

    def add_item(self, name, price):
        if not isinstance(price, (int, float)):
            raise ValueError("Price must be numeric")
        self.items.append((name, price))

    def remove_item(self, name):
        for i, (item_name, price) in enumerate(self.items):
            if item_name == name:
                self.items.pop(i)
                return
        # If not found, ignore (graceful handling)

    def total_cost(self):
        return sum(price for _, price in self.items)

class TestShoppingCart(unittest.TestCase):
    def setUp(self):
        self.cart = ShoppingCart()

    def test_empty_cart_total(self):
        self.assertEqual(self.cart.total_cost(), 0)
```

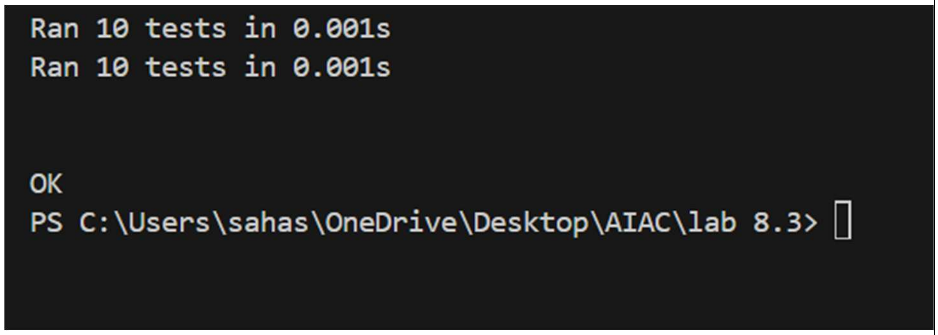


```python
    def test_add_single_item(self):
        self.cart.add_item("Apple", 1.5)
        self.assertEqual(self.cart.total_cost(), 1.5)

    def test_add_multiple_items(self):
        self.cart.add_item("Apple", 1.5)
        self.cart.add_item("Banana", 2.0)
        self.assertEqual(self.cart.total_cost(), 3.5)

    def test_remove_item(self):
        self.cart.add_item("Apple", 1.5)
        self.cart.add_item("Banana", 2.0)
        self.cart.remove_item("Apple")
        self.assertEqual(self.cart.total_cost(), 2.0)

    def test_remove_nonexistent_item(self):
        self.cart.add_item("Apple", 1.5)
        self.cart.remove_item("Water")  # should not raise error
        self.assertEqual(self.cart.total_cost(), 1.5)

    def test_add_duplicate_items(self):
        self.cart.add_item("Apple", 1.5)
        self.cart.add_item("Apple", 1.5)
        self.assertEqual(self.cart.total_cost(), 3.0)

    def test_remove_one_of_duplicates(self):
        self.cart.add_item("Apple", 1.5)
```



```
        self.cart.add_item("Apple", 1.5)
        self.cart.remove_item("Apple")
        self.assertEqual(self.cart.total_cost(), 1.5)

    def test_add_zero_price_item(self):
        self.cart.add_item("Coupon", 0)
        self.assertEqual(self.cart.total_cost(), 0)

    def test_add_negative_price_item(self):
        self.cart.add_item("Discount", -5)
        self.assertEqual(self.cart.total_cost(), -5)

    def test_empty_cart_after_removals(self):
        self.cart.add_item("Apple", 1.5)
        self.cart.remove_item("Apple")
        self.assertEqual(self.cart.total_cost(), 0)

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

```

**OUTPUT:**



```
Ran 10 tests in 0.001s
Ran 10 tests in 0.001s


OK
PS C:\Users\sahas\OneDrive\Desktop\AIAC\lab 8.3> 
```

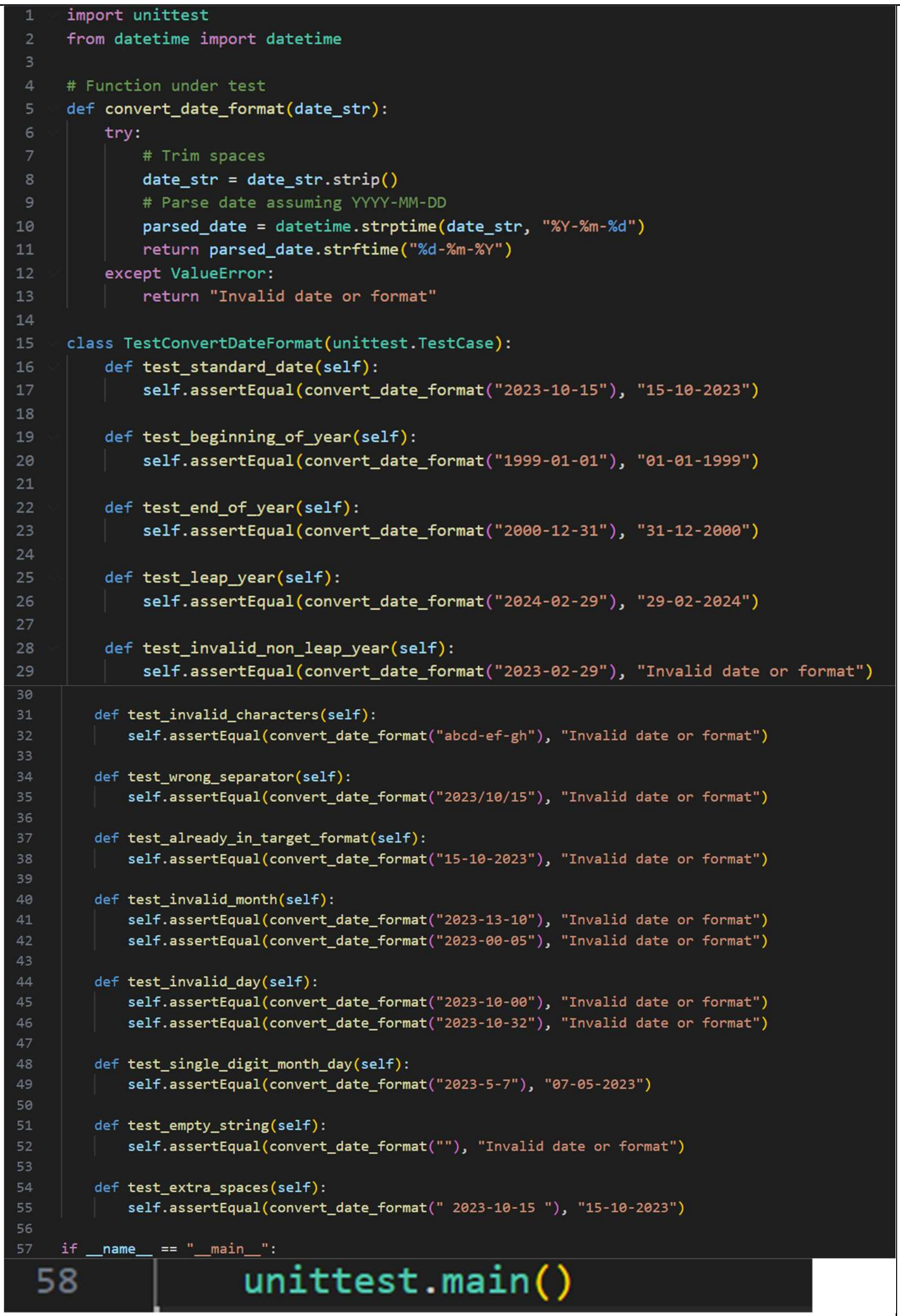
**Task Description#5**

- Use AI to write test cases for convert_date_format(date_str) to switch from "YYYY-MM-DD" to "DD-MM-YYYY".
  **Example: "2023-10-15" → "15-10-2023"**

**Expected Output#5**

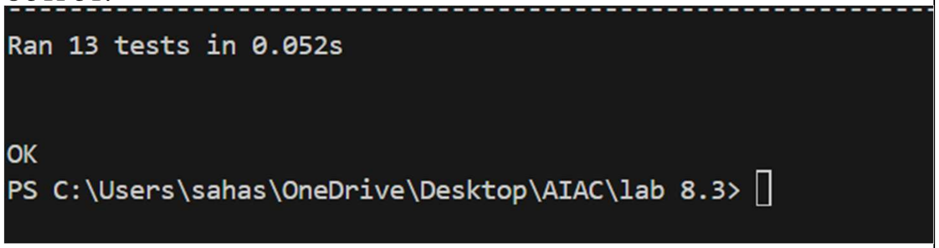
- Function converts input format correctly for all test cases

CODE:



```python
import unittest
from datetime import datetime

# Function under test
def convert_date_format(date_str):
    try:
        # Trim spaces
        date_str = date_str.strip()
        # Parse date assuming YYYY-MM-DD
        parsed_date = datetime.strptime(date_str, "%Y-%m-%d")
        return parsed_date.strftime("%d-%m-%Y")
    except ValueError:
        return "Invalid date or format"

class TestConvertDateFormat(unittest.TestCase):
    def test_standard_date(self):
        self.assertEqual(convert_date_format("2023-10-15"), "15-10-2023")

    def test_beginning_of_year(self):
        self.assertEqual(convert_date_format("1999-01-01"), "01-01-1999")

    def test_end_of_year(self):
        self.assertEqual(convert_date_format("2000-12-31"), "31-12-2000")

    def test_leap_year(self):
        self.assertEqual(convert_date_format("2024-02-29"), "29-02-2024")

    def test_invalid_non_leap_year(self):
        self.assertEqual(convert_date_format("2023-02-29"), "Invalid date or format")

    def test_invalid_characters(self):
        self.assertEqual(convert_date_format("abcd-ef-gh"), "Invalid date or format")

    def test_wrong_separator(self):
        self.assertEqual(convert_date_format("2023/10/15"), "Invalid date or format")

    def test_already_in_target_format(self):
        self.assertEqual(convert_date_format("15-10-2023"), "Invalid date or format")

    def test_invalid_month(self):
        self.assertEqual(convert_date_format("2023-13-10"), "Invalid date or format")
        self.assertEqual(convert_date_format("2023-00-05"), "Invalid date or format")

    def test_invalid_day(self):
        self.assertEqual(convert_date_format("2023-10-00"), "Invalid date or format")
        self.assertEqual(convert_date_format("2023-10-32"), "Invalid date or format")

    def test_single_digit_month_day(self):
        self.assertEqual(convert_date_format("2023-5-7"), "07-05-2023")

    def test_empty_string(self):
        self.assertEqual(convert_date_format(""), "Invalid date or format")

    def test_extra_spaces(self):
        self.assertEqual(convert_date_format(" 2023-10-15 "), "15-10-2023")

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

**OUTPUT:**



```
----------------------------------------------------------------------
Ran 13 tests in 0.052s


OK
PS C:\Users\sahas\OneDrive\Desktop\AIAC\lab 8.3> 
```

**Note: Report should be submitted a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots**

**Evaluation Criteria:**

| Criteria | Max Marks |
|---|---|
| Task #1 | 0.5 |
| Task #2 | 0.5 |
| Task #3 | 0.5 |
| Task #4 | 0.5 |
| Task #5 | 0.5 |
| **Total** | **2.5 Marks** |