AI-ASSISSTANT-CODING-LAB-8.3

Name: V.Abhinav

Batch:41

Roll-No:2303A52174

Task 1: Email Validation using TDD

CODE:

```
# TASK 1: EMAIL VALIDATION
# ============================================================

# AI PROMPT:
# Generate unittest test cases for email validation function.
# Requirements:
# - Must contain @ and .
# - Must not start or end with special characters
# - Should not allow multiple @ symbols
# Include valid and invalid cases.


def is_valid_email(email):
    if not isinstance(email, str):
        return False

    if email.count("@") != 1:
        return False

    if "@" not in email or "." not in email:
        return False

    if not email[0].isalnum() or not email[-1].isalnum():
        return False

    pattern = r'^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$'
    return bool(re.match(pattern, email))
```

```python
class TestEmailValidation(unittest.TestCase):

    def test_valid_emails(self):
        self.assertTrue(is_valid_email("user@example.com"))
        self.assertTrue(is_valid_email("john.doe@gmail.com"))

    def test_invalid_emails(self):
        self.assertFalse(is_valid_email("userexample.com"))
        self.assertFalse(is_valid_email("user@@example.com"))
        self.assertFalse(is_valid_email("@example.com"))
        self.assertFalse(is_valid_email("user@example.com."))
```

OUTPUT:

```
==========================================
          TASK 1: EMAIL VALIDATION OUTPUT
==========================================
test_invalid_emails (__main__.TestEmailValidation.test_invalid_emails) ... ok
test_valid_emails (__main__.TestEmailValidation.test_valid_emails) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.001s
```

Justification:

The email validation task was implemented using a Test-Driven Development (TDD) approach. AI-generated test cases were created first to define valid and invalid email behaviors. This ensured that the validation logic strictly followed the requirements before writing the actual function.

Task 2: Grade Assignment using Loops

Scenario

You are building an automated grading system for an online examination platform.

Requirements

• AI should generate test cases for assign_grade(score) where:
– 90–100 → A
– 80–89 → B
– 70–79 → C
– 60–69 → D
– Below 60 → F
• Include boundary values (60, 70, 80, 90)
• Include invalid inputs such as -5, 105, "eighty"
• Implement the function using a test-driven approach

Prompt: Generate unittest test cases for a Python function named assign_grade(score**).**

CODE:

```
# TASK 2: GRADE ASSIGNMENT
# ============================================================

# AI PROMPT:
# Generate unittest test cases for assign_grade(score)
# Include boundary values (60,70,80,90)
# Include invalid inputs (-5,105,"eighty")


def assign_grade(score):

    if not isinstance(score, int):
        return "Invalid"

    if score < 0 or score > 100:
        return "Invalid"

    if score >= 90:
        return "A"
    elif score >= 80:
        return "B"
    elif score >= 70:
        return "C"
    elif score >= 60:
        return "D"
    else:
        return "F"
```

```python
class TestGradeAssignment(unittest.TestCase):

    def test_grades(self):
        self.assertEqual(assign_grade(95), "A")
        self.assertEqual(assign_grade(90), "A")
        self.assertEqual(assign_grade(85), "B")
        self.assertEqual(assign_grade(80), "B")
        self.assertEqual(assign_grade(75), "C")
        self.assertEqual(assign_grade(70), "C")
        self.assertEqual(assign_grade(65), "D")
        self.assertEqual(assign_grade(60), "D")
        self.assertEqual(assign_grade(50), "F")

    def test_invalid(self):
        self.assertEqual(assign_grade(-5), "Invalid")
        self.assertEqual(assign_grade(105), "Invalid")
        self.assertEqual(assign_grade("eighty"), "Invalid")
```

OUTPUT:

```
==============================================
          TASK 2: GRADE ASSIGNMENT OUTPUT
==============================================
test_grades (__main__.TestGradeAssignment.test_grades) ... ok
test_invalid (__main__.TestGradeAssignment.test_invalid) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.000s
```

Justification:
The grade assignment system was developed using TDD to ensure correct grade mapping and boundary handling. AI-generated test cases helped identify important boundary values such as 60, 70, 80, and 90, which are critical in grading systems.

Task 3: Sentence Palindrome Checker
Scenario
You are developing a text-processing utility to analyze sentences.
Requirements
• AI should generate test cases for is_sentence_palindrome(sentence)
• Ignore case, spaces, and punctuation
• Test both palindromic and non-palindromic sentences
• Example: "A man a plan a canal Panama" → True

Prompt: Generate unittest test cases for a Python function named is_sentence_palindrome(sentence).This function checks whether a sentence is a palindrome.

CODE:

```
# TASK 3: SENTENCE PALINDROME CHECKER
# ============================================================
# AI PROMPT:
# Generate unittest test cases for sentence palindrome checker.
# Ignore case, spaces, punctuation.
# Include palindromes and non-palindromes.
def is_sentence_palindrome(sentence):

    if not isinstance(sentence, str):
        return False

    cleaned = ''.join(char.lower() for char in sentence if char.isalnum())
    return cleaned == cleaned[::-1]


class TestPalindrome(unittest.TestCase):

    def test_palindromes(self):
        self.assertTrue(is_sentence_palindrome("A man a plan a canal Panama"))
        self.assertTrue(is_sentence_palindrome("Was it a car or a cat I saw?"))

    def test_non_palindromes(self):
        self.assertFalse(is_sentence_palindrome("Hello world"))
        self.assertFalse(is_sentence_palindrome("Python is fun"))
```

OUTPUT:

```
============================================
          TASK 3: PALINDROME CHECKER OUTPUT
============================================
test_non_palindromes (__main__.TestPalindrome.test_non_palindromes) ... ok
test_palindromes (__main__.TestPalindrome.test_palindromes) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.000s
```

Justification:
This task required ignoring case sensitivity, spaces, and punctuation. AI-generated test cases ensured that both simple and complex palindromes were tested before implementation.

Task 4: ShoppingCart Class

Scenario

You are designing a basic shopping cart module for an e-commerce application.

Requirements

• AI should generate test cases for the ShoppingCart class
• Class must include the following methods:
– add_item(name, price)
– remove_item(name)
– total_cost()
• Validate correct addition, removal, and cost calculation
• Handle empty cart scenarios

Prompt: Generate unittest test cases for a Python class named ShoppingCart.
The class must include the following methods:

- add_item(name, price)
- remove_item(name)
- total_cost()

CODE:

```
# ============================================================
# TASK 4: SHOPPING CART CLASS
# ============================================================

# AI PROMPT:
# Generate unittest test cases for ShoppingCart class.
# Methods:
# - add_item(name, price)
# - remove_item(name)
# - total_cost()
# Test empty cart, add, remove, cost.


class ShoppingCart:

    def __init__(self):
        self.items = {}

    def add_item(self, name, price):
        if price < 0:
            return
        self.items[name] = price

    def remove_item(self, name):
        if name in self.items:
            del self.items[name]

    def total_cost(self):
        return sum(self.items.values())
```

```python
class TestShoppingCart(unittest.TestCase):

    def test_add_and_total(self):
        cart = ShoppingCart()
        cart.add_item("Pen", 20)
        cart.add_item("Book", 80)
        self.assertEqual(cart.total_cost(), 100)

    def test_remove_item(self):
        cart = ShoppingCart()
        cart.add_item("Pen", 20)
        cart.remove_item("Pen")
        self.assertEqual(cart.total_cost(), 0)

    def test_empty_cart(self):
        cart = ShoppingCart()
        self.assertEqual(cart.total_cost(), 0)
```

OUTPUT:

```
==============================================
          TASK 4: SHOPPING CART OUTPUT
==============================================
test_add_and_total (__main__.TestShoppingCart.test_add_and_total) ... ok
test_empty_cart (__main__.TestShoppingCart.test_empty_cart) ... ok
test_remove_item (__main__.TestShoppingCart.test_remove_item) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.000s
```

Justification:
The ShoppingCart class was designed using AI-generated test cases to validate object behavior before full implementation. TDD ensured that all methods functioned correctly under different scenarios.

Task 5: Date Format Conversion
Scenario
You are creating a utility function to convert date formats for reports.
Requirements
• AI should generate test cases for convert_date_format(date_str)
• Input format must be "YYYY-MM-DD"
• Output format must be "DD-MM-YYYY"
• Example: "2023-10-15" → "15-10-2023"
Prompt: Generate unittest test cases for a Python function named convert_date_format(date_str).This function converts date format from "YYYY-MM-DD" to "DD-MM-YYYY".

CODE:

```
# TASK 5: DATE FORMAT CONVERSION
# ============================================================

# AI PROMPT:
# Generate unittest test cases for convert_date_format(date_str)
# Input: YYYY-MM-DD
# Output: DD-MM-YYYY
# Include valid and invalid cases.


def convert_date_format(date_str):
    try:
        date_obj = datetime.strptime(date_str, "%Y-%m-%d")
        return date_obj.strftime("%d-%m-%Y")
    except ValueError:
        return "Invalid"


class TestDateConversion(unittest.TestCase):

    def test_valid_dates(self):
        self.assertEqual(convert_date_format("2023-10-15"), "15-10-2023")
        self.assertEqual(convert_date_format("2024-01-01"), "01-01-2024")

    def test_invalid_dates(self):
        self.assertEqual(convert_date_format("15-10-2023"), "Invalid")
        self.assertEqual(convert_date_format("2023/10/15"), "Invalid")
```

OUTPUT:

```
==========================================
          TASK 5: DATE FORMAT CONVERSION OUTPUT
==========================================
test_invalid_dates (__main__.TestDateConversion.test_invalid_dates) ... ok
test_valid_dates (__main__.TestDateConversion.test_valid_dates) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.006s
```

Justification:

The date conversion utility required strict format validation. AI-generated test cases helped verify correct conversion from "YYYY-MM-DD" to "DD-MM-YYYY" and ensured invalid formats were handled properly.