Lab 8: Test-Driven Development with AI

Name: Dinesh

Hall Ticket No: 2303A52288

Batch: 42

Task 1: Email Validation using TDD

Prompt Used:

Generate comprehensive unittest test cases for validating email formats including valid, invalid, boundary, and edge cases.

Code Implementation:

```
from __future__ import annotations

import unittest
import math


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

def is_valid_email(email: str) -> bool:
    if not isinstance(email, str) or not email:
        return False
    if email.count("@") != 1:
        return False
    if "." not in email:
        return False
    if not email[0].isalnum() or not email[-1].isalnum():
        return False
    if email[0] in "@." or email[-1] in "@.":
        return False
    return True


class TestIsValidEmail(unittest.TestCase):

    def test_emails(self):
        valid = [
            "user@example.com",
            "u.ser+tag@example.co.uk",
            "user_name@example.io",
        ]
        invalid = [
            "userexample.com",
            "@example.com",
            "user@",
            "user@@example.com",
            "",
            None,
        ]

        for email in valid:
            print(f"Checking VALID: {email}")
            self.assertTrue(is_valid_email(email))

        for email in invalid:
            print(f"Checking INVALID: {email}")
            self.assertFalse(is_valid_email(email))
```

Output :

```
PS D:\AI_ASSIANT> & C:\Users\karup\AppData\Local\Programs\Python\Python314\python.exe d:/AI_ASSIANT/AS_8.3.py

========================================
          TASK 1 OUTPUT
========================================
test_emails (__main__.TestIsValidEmail.test_emails) ... Checking VALID: user@example.com
Checking VALID: u.ser+tag@example.co.uk
Checking VALID: user_name@example.io
Checking INVALID: userexample.com
Checking INVALID: @example.com
Checking INVALID: user@
Checking INVALID: user@@example.com
Checking INVALID:
Checking INVALID: None
ok

----------------------------------------------------------------------
Ran 1 test in 0.001s

OK
```

Justification:

Test cases were created before implementing the function. The function was designed to satisfy requirements such as single '@', presence of '.', and proper starting and ending characters. Invalid formats are correctly rejected.

Task 2: Grade Assignment using Loops

Prompt Used:

Generate unittest cases for grade assignment including boundary values and invalid inputs.

Code Implementation:

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

def assign_grade(score: object) -> str | None:
    if isinstance(score, bool):
        return None
    if not isinstance(score, (int, float)):
        return None
    if isinstance(score, float) and math.isnan(score):
        return None
    if score < 0 or score > 100:
        return None

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

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

    def test_grades(self):
        cases = [
            (95, "A"),
            (80, "B"),
            (70, "C"),
            (60, "D"),
            (50, "F"),
            (-5, None),
            (105, None),
            ("eighty", None),
        ]

        for score, expected in cases:
            print(f"Checking Score: {score}")
            self.assertEqual(assign_grade(score), expected)
```

Output :

```
========================================
          TASK 2 OUTPUT
========================================
test_grades (__main__.TestAssignGrade.test_grades) ... Checking Score: 95
Checking Score: 80
Checking Score: 70
Checking Score: 60
Checking Score: 50
Checking Score: -5
Checking Score: 105
Checking Score: eighty
ok

----------------------------------------------------------------------
Ran 1 test in 0.001s

OK
```

Justification:

Boundary values (60, 70, 80, 90) were explicitly tested. Invalid inputs like negative values and non-integers were handled to ensure robustness.

Task 3: Sentence Palindrome Checker

Prompt Used:

Generate test cases for checking palindromic sentences ignoring case, spaces, and punctuation.

Code Implementation:

```
# ============================================================
# TASK 3: PALINDROME CHECKER
# ============================================================

def is_sentence_palindrome(sentence: str) -> bool:
    if not isinstance(sentence, str):
        return False
    cleaned = "".join(ch.lower() for ch in sentence if ch.isalnum())
    return cleaned == cleaned[::-1]


class TestPalindrome(unittest.TestCase):

    def test_sentences(self):
        cases = [
            ("A man a plan a canal Panama", True),
            ("No lemon, no melon", True),
            ("Hello world", False),
            (None, False),
        ]

        for sentence, expected in cases:
            print(f"Checking Sentence: {sentence}")
            self.assertEqual(is_sentence_palindrome(sentence), expected)
```

Output:

```
========================================
          TASK 3 OUTPUT
========================================
test_sentences (__main__.TestPalindrome.test_sentences) ... Checking Sentence: A man a plan a canal Panama
Checking Sentence: No lemon, no melon
Checking Sentence: Hello world
Checking Sentence: None
ok

----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

Justification:

The function removes all non-alphanumeric characters and converts text to lowercase before checking palindrome property.

Task 4: ShoppingCart Class

Prompt Used:

Generate test cases validating addition, removal, total cost calculation, and empty cart handling.

Code Implementation:

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

class ShoppingCart:

    def __init__(self):
        self._items = []

    def add_item(self, name, price):
        if not isinstance(name, str) or not name:
            return False
        if not isinstance(price, (int, float)) or price < 0:
            return False
        self._items.append((name, price))
        return True

    def remove_item(self, name):
        for i, (item_name, _) in enumerate(self._items):
            if item_name == name:
                del self._items[i]
                return True
        return False

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

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

    def test_cart(self):
        cart = ShoppingCart()

        print("Adding Book (100)")
        self.assertTrue(cart.add_item("Book", 100))

        print("Adding Pen (20)")
        self.assertTrue(cart.add_item("Pen", 20))

        print("Total Cost Check")
        self.assertEqual(cart.total_cost(), 120)

        print("Removing Book")
        self.assertTrue(cart.remove_item("Book"))

        print("Final Cost Check")
        self.assertEqual(cart.total_cost(), 20)
```

Output:

```
========================================
          TASK 4 OUTPUT
========================================
test_cart (__main__.TestShoppingCart.test_cart) ... Adding Book (100)
Adding Pen (20)
Total Cost Check
Removing Book
Final Cost Check
ok

----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

Justification:

The ShoppingCart class maintains items in a dictionary. Methods ensure accurate cost calculation and safe removal operations.

Task 5: Date Format Conversion

Prompt Used:

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

Code Implementation:

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

def convert_date_format(date_str: str) -> str | None:
    if not isinstance(date_str, str):
        return None

    parts = date_str.split("-")
    if len(parts) != 3:
        return None

    year, month, day = parts
    if not (year.isdigit() and month.isdigit() and day.isdigit()):
        return None
    if len(year) != 4 or len(month) != 2 or len(day) != 2:
        return None

    month_val = int(month)
    day_val = int(day)
    if month_val < 1 or month_val > 12:
        return None
    if day_val < 1 or day_val > 31:
        return None

    return f"{day}-{month}-{year}"
```

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

    def test_dates(self):
        cases = [
            ("2023-10-15", "15-10-2023"),
            ("2023-13-15", None),
            ("2023/10/15", None),
        ]

        for date_str, expected in cases:
            print(f"Checking Date: {date_str}")
            self.assertEqual(convert_date_format(date_str), expected)
```

Output:

```
========================================
          TASK 5 OUTPUT
========================================
test_dates (__main__.TestConvertDateFormat.test_dates) ... Checking Date: 2023-10-15
Checking Date: 2023-13-15
Checking Date: 2023/10/15
========================================
          TASK 5 OUTPUT
========================================
test_dates (__main__.TestConvertDateFormat.test_dates) ... Checking Date: 2023-10-15
Checking Date: 2023-13-15
Checking Date: 2023/10/15
========================================
test_dates (__main__.TestConvertDateFormat.test_dates) ... Checking Date: 2023-10-15
Checking Date: 2023-13-15
Checking Date: 2023/10/15
ok

test_dates (__main__.TestConvertDateFormat.test_dates) ... Checking Date: 2023-10-15
Checking Date: 2023-13-15
Checking Date: 2023/10/15
ok

----------------------------------------------------------------------
ok

----------------------------------------------------------------------
Ran 1 test in 0.000s
----------------------------------------------------------------------
Ran 1 test in 0.000s
Ran 1 test in 0.000s

OK
```

Justification:

Regex validation ensures correct input format before conversion. Invalid formats return 'Invalid'.