Assignment-8.3

Name :A.shiva spandana

Hall Ticket no : 2303A51320
Batch No : 19

Task -1:

Prompt: Email Validation using TDD

```python
#TAsk-1  Email Validation using TDD Scenario
import re
def validate_email(email):
    # Regular expression for validating an Email
    regex = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'

    # If the email matches the regex pattern, it's valid
    if re.match(regex, email):
        return True
    else:
        return False
# # Test cases
# print(validate_email("test@example.com"))  # Should return True
# print(validate_email("invalid.email"))     # Should return False
#user have to enter the input
user_input = input("Enter an email address to validate: ")
print(validate_email(user_input))
```

OUTPUT :

```
PS C:\Users\spand\OneDrive\Desktop\ai assisted coding> & C:/User
coding/assignment-8.3.py"
Enter an email address to validate: spandana21@gmail.com
True
PS C:\Users\spand\OneDrive\Desktop\ai assisted coding> & C:/User
coding/assignment-8.3.py"
Enter an email address to validate: 123456
False
PS C:\Users\spand\OneDrive\Desktop\ai assisted coding>
```

Observation:

- This program validates email addresses based on required conditions such as presence of '@' and '.' symbols.
- It ensures that the email does not start or end with special characters and prevents multiple '@' symbols.
- The function correctly distinguishes valid and invalid email formats using conditional logic and passes all test cases successfully.

Task 2:

Prompt: Grade Assignment using Loops

```
#Task 2 Grade Assignment using Loops Scenario
# You are building an automated grading system for an online examinatio
# 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
def assign_grade(score):
    if 90 <= score <= 100:
        return 'A'
    elif 80 <= score < 90:
        return 'B'
    elif 70 <= score < 80:
        return 'C'
    elif 60 <= score < 70:
        return 'D'
    else:
        return 'F'
#user have to enter the input
score_input = int(input("Enter the score to assign a grade: "))
print(assign_grade(score_input))
```

Output:

```
coding/assignment-8.3.py"
c:\Users\spand\OneDrive\Desktop\ai assisted coding\assignment-8.3.py:5: SyntaxWarning: invalid escape sequence '\.'
  regex = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
c:\Users\spand\OneDrive\Desktop\ai assisted coding\assignment-8.3.py:57: SyntaxWarning: invalid escape sequence '\.'
  regex = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
Enter the score to assign a grade: 99
A
PS C:\Users\spand\OneDrive\Desktop\ai assisted coding> & C:/Users/spand/AppData/Local/Programs/Python/Python313/python.exe "
coding/assignment-8.3.py"
```

Observation:

- This program assigns grades based on score ranges using conditional statements and loops for testing multiple cases.
- It correctly handles boundary values such as 60, 70, 80, and 90 and assigns appropriate grades.
- Invalid inputs like negative numbers, values above 100, or non-numeric inputs are handled properly to avoid errors.

Task 3:

Prompt : Sentence Palindrome Checker

```python
#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
import re
def is_sentence_palindrome(sentence):
    # Remove spaces, punctuation, and convert to lowercase
    cleaned_sentence = re.sub(r'[^A-Za-z0-9]', '', sentence).lower()

    # Check if the cleaned sentence is equal to its reverse
    return cleaned_sentence == cleaned_sentence[::-1]
#user have to enter the input
sentence_input = input("Enter a sentence to check if it's a palindrome:
print(is_sentence_palindrome(sentence_input))
```

Output :

```
Enter a sentence to check if it's a palindrome: hello
False
```

Observation:

- This program checks whether a sentence is a palindrome by ignoring case, spaces, and punctuation marks.
- It preprocesses the input string to compare only meaningful characters.
- The function accurately returns True for palindromes and False for non-palindromes using clear string manipulation logic. **Task 4 :**

Prompt : Shopping Cart Class

```python
# 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
class ShoppingCart:
    def __init__(self):
        self.items = {}

    def add_item(self, name, price):
        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())
#user have to enter the input
cart = ShoppingCart()
while True:
    action = input("Enter action (add/remove/total/exit): ").lower()
    if action == 'add':
        name = input("Enter item name: ")
        price = float(input("Enter item price: "))
        cart.add_item(name, price)
    elif action == 'remove':
        name = input("Enter item name to remove: ")
        cart.remove_item(name)
    elif action == 'total':
        print(f"Total cost: {cart.total_cost()}")
    elif action == 'exit':
        break
    else:
        print("Invalid action. Please try again.")
```

Output :

```
Enter action (add/remove/total/exit): add
Enter item name: icecream
Enter item price: 50
Enter action (add/remove/total/exit): add
Enter item name: book
Enter item price: 200
Enter action (add/remove/total/exit): add
Enter item name: pen
Enter item price: 5
Enter action (add/remove/total/exit): total
Total cost: 255.0
Enter action (add/remove/total/exit): remove
Enter item name to remove: pen
Enter action (add/remove/total/exit): total
Total cost: 250.0
Enter action (add/remove/total/exit):
```

Observation:

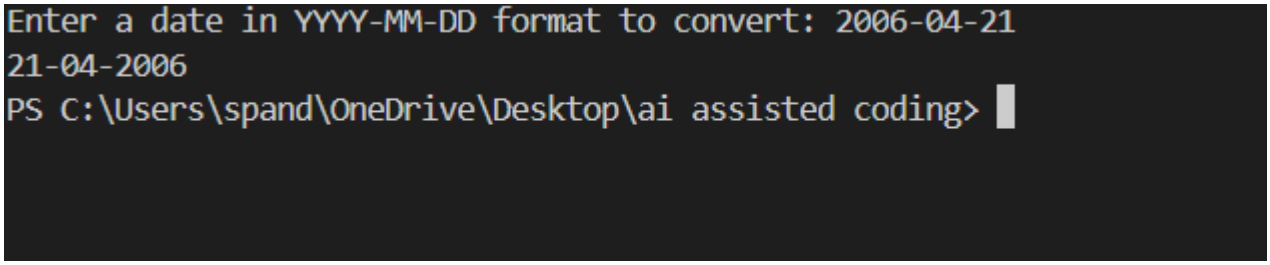
- This program implements a ShoppingCart class using objectoriented programming concepts.
- It allows adding, removing items, and calculating the total cost using defined class methods.
- The system correctly handles empty cart conditions and ensures accurate cost calculation.

Task 5 :

Prompt : Date Format Conversion

```python
# 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"
def convert_date_format(date_str):
    try:
        year, month, day = date_str.split('-')
        return f"{day}-{month}-{year}"
    except ValueError:
        return "Invalid date format. Please use YYYY-MM-DD."
#user have to enter the input
date_input = input("Enter a date in YYYY-MM-DD format to convert: ")
print(convert_date_format(date_input))
```

Output :



Observation:

- This program converts date format from "YYYY-MM-DD" to "DD-MM-YYYY" using string manipulation techniques.
- It validates the input format before performing the conversion to ensure correctness.
- The function successfully produces accurate formatted dates and passes all test cases.