Assignment 8.1

2303A51243

K.Pallavi

Batch – 18

Task Description #1: (Password Strength Validator – Apply AI in Security Context)

• Task: Apply AI to generate at least 3 assert test cases for is_strong_password(password) and implement the validator function.

• Requirements:

o Password must have at least 8 characters.

o Must include uppercase, lowercase, digit, and special character. o Must not contain spaces. Example Assert Test Cases:

assert is_strong_password("Abcd@123") == True

assert is_strong_password("abcd123") == False

assert is_strong_password("ABCD@1234") == True

Expected Output #1:

• Password validation logic passing all AI-generated test cases.

Prompt :

#Create a Python function is_strong_password(password) that validates password strength.
#The password must:
#Have at least 8 characters
#Contain at least one uppercase letter
#Contain at least one lowercase letter
#Contain at least one digit
#Contain at least one special character
#Must not contain spaces

#Generate at least 3 assert test cases to verify the function.

password.py

```python
#The password must:
#Have at least 8 characters
#Contain at least one uppercase letter
#Contain at least one lowercase letter
#Contain at least one digit
#Contain at least one special character
#Must not contain spaces
#Generate at least 3 assert test cases to verify the function.
import string
def is_strong_password(password):
    if len(password) < 8:
        return False

    if " " in password:
        return False
    has_upper = any(char.isupper() for char in password)
    has_lower = any(char.islower() for char in password)
    has_digit = any(char.isdigit() for char in password)
    has_special = any(char in string.punctuation for char in password)
    return has_upper and has_lower and has_digit and has_special
# Assert Test Cases
assert is_strong_password("Abcd@123") == True
assert is_strong_password("abcd123") == False
assert is_strong_password("ABCD@1234") == False
assert is_strong_password("Abc 123@") == False
assert is_strong_password("Strong@9") == True
print("All test cases passed successfully!")
```

Output

```
PS C:\Users\ANIL\Desktop\AI Assisant> & C:/Users/ANIL/AppData/Local/Python/pythoncore-3.14-64/python.exe "c:/Users/ANIL/Desktop/AI Assisant/password.py"
All test cases passed successfully!
PS C:\Users\ANIL\Desktop\AI Assisant>
```

Analysis

1.The password validator checks multiple security conditions.

2. The AI-generated assert test cases helped verify both valid and invalid scenarios.

3. no assertion errors occurred, the function works correctly.

Task Description #2: (Number Classification with Loops – Apply AI for Edge Case Handling)

• Task: Use AI to generate at least 3 assert test cases for a classify_number(n) function. Implement using loops.

• Requirements:

o Classify numbers as Positive, Negative, or Zero. o Handle invalid inputs like strings and None. o Include boundary conditions (-1, 0, 1).

Example Assert Test Cases:

assert classify_number(10) == "Positive"

assert classify_number(-5) == "Negative"

assert classify_number(0) == "Zero"

Expected Output #2:

• Classification logic passing all assert tests.

Prompt :

#Number Classification with Loops classify_number(n) function. Implement using loops.

#Classify numbers as Positive, Negative, or Zero.

Handle invalid inputs like strings and None.

Include boundary conditions (-1, 0, 1).

Test cases from user input

Code:

numberclassification.py > ...

```python
#Number Classification with Loops  classify_number(n) function. Implement using loops.
#Classify numbers as Positive, Negative, or Zero.
# Handle invalid inputs like strings and None.
# Include boundary conditions (-1, 0, 1).
def classify_number(n):
    if n is None:
        return "Invalid input: None"
    if isinstance(n, str):
        return "Invalid input: String"

    if n > 0:
        return "Positive"
    elif n < 0:
        return "Negative"
    else:
        return "Zero"
# Test cases from user input
print(classify_number(10))  # Positive
print(classify_number(-5))  # Negative
print(classify_number(0))   # Zero
print(classify_number(-1))  # Negative
print(classify_number(1))   # Positive
print(classify_number("Hello"))  # Invalid input: String
print(classify_number(None))  # Invalid input: None
print("All test cases executed successfully!")
```

Output:

```
PS C:\Users\ANIL\Desktop\AI Assisant> & C:/Users/ANIL/AppData/Local/Python/pythoncore-3.14-64/python.exe "c:/Users/ANIL/Desktop/
Positive
Negative
Zero
Negative
Positive
Invalid input: String
Invalid input: None
All test cases executed successfully!
PS C:\Users\ANIL\Desktop\AI Assisant>
```

Analysis:

1. The function classifies numbers into Positive, Negative, or Zero.
2. Invalid inputs like strings and None return "Invalid Input".
3. AI-generated assert cases helped verify normal cases, edge cases, and invalid inputs.

Task Description #3: (Anagram Checker – Apply AI for String Analysis)

• Task: Use AI to generate at least 3 assert test cases for is_anagram(str1, str2) and implement the function.

• Requirements:

o Ignore case, spaces, and punctuation. o Handle edge cases (empty strings, identical words).

Example Assert Test Cases:

assert is_anagram("listen", "silent") == True assert is_anagram("hello", "world") == False assert is_anagram("Dormitory", "Dirty Room") == True

Expected Output #3:

• Function correctly identifying anagrams and passing all AI-generated tests.

Prompt :

#Anagram Checker least 3 assert test cases foris_anagram(str1, str2) and implement the function.

#Ignore case, spaces, and punctuation.

Handle edge cases (empty strings, identical words).

Test cases from user input

anagramchecker.py > ...

```python
#Anagram Checker least 3 assert test cases foris_anagram(str1, str2) and implement the function.
#Ignore case, spaces, and punctuation.
# Handle edge cases (empty strings, identical words).
def is_anagram(str1, str2):
    # Remove non-alphabetic characters and convert to lowercase
    clean_str1 = ''.join(c.lower() for c in str1 if c.isalpha())
    clean_str2 = ''.join(c.lower() for c in str2 if c.isalpha())

    # If lengths are different, they can't be anagrams
    if len(clean_str1) != len(clean_str2):
        return False

    # If strings are identical (after cleaning), they are anagrams
    if clean_str1 == clean_str2:
        return True

    # Sort characters and compare
    return sorted(clean_str1) == sorted(clean_str2)

# Test cases from user input
assert is_anagram("Listen", "Silent") == True
assert is_anagram("Triangle", "Integral") == True
assert is_anagram("Apple", "Pabble") == False
assert is_anagram("Dormitory", "Dirty Room") == True
assert is_anagram("A gentleman", "Elegant man") == True
assert is_anagram("", "") == True
print("All test cases passed successfully!")
```

Output:

```
PS C:\Users\ANIL\Desktop\AI Assisant> & C:/Users/ANIL/AppData/Local/Python/pythoncore-3.14-64/python.exe "c:/Users/AN
All test cases passed successfully!
PS C:\Users\ANIL\Desktop\AI Assisant>
```

Analysis:

1. The function checks if two strings are anagrams.
2. AI-generated assert test cases verified both valid and invalid scenarios.
3. Since no assertion errors occurred, the function works correctly.

Task Description #4:

(Inventory Class – Apply AI to Simulate Real-

World Inventory System)

• Task: Ask AI to generate at least 3 assert-based tests for an Inventory class with stock management.

• Methods:

o add_item(name, quantity) o

remove_item(name, quantity) o

get_stock(name) Example Assert

Test Cases: inv = Inventory()

inv.add_item("Pen", 10) assert

inv.get_stock("Pen") == 10

inv.remove_item("Pen", 5) assert

inv.get_stock("Pen") == 5

inv.add_item("Book", 3) assert

inv.get_stock("Book") == 3

Expected Output #4:

• Fully functional class passing all assertions.

Prompt :

#The class must include:

#add_item(name, quantity)

#remove_item(name, quantity)

#get_stock(name) #Requirements:

#Prevent removing more stock

than available.

#Handle items that do not exist.

#Generate at least 3 assert-based test cases to validate functionality.

Code :

inventoryclass.py > ...

```python
#Create a Python class Inventory that simulates a real-world stock management system.
#The class must include:
#add_item(name, quantity)
#remove_item(name, quantity)
#get_stock(name)
#Requirements:
#Prevent removing more stock than available.
#Handle items that do not exist.
#Generate at least 3 assert-based test cases to validate functionality.
class Inventory:
    def __init__(self):
        self.stock = {}

    def add_item(self, name, quantity):
        if name in self.stock:
            self.stock[name] += quantity
        else:
            self.stock[name] = quantity

    def remove_item(self, name, quantity):
        if name not in self.stock:
            return "Item does not exist"
        if self.stock[name] < quantity:
            return "Not enough stock to remove"
        self.stock[name] -= quantity
        return "Item removed successfully"
```

```python
inventoryclass.py > ...
class Inventory:
    def remove_item(self, name, quantity):
        return "Item removed successfully"

    def get_stock(self, name):
        return self.stock.get(name, "Item does not exist")
# Assert Test Cases
inventory = Inventory()
inventory.add_item("Apple", 50)
inventory.add_item("Banana", 30)
assert inventory.get_stock("Apple") == 50
assert inventory.get_stock("Banana") == 30
assert inventory.get_stock("Orange") == "Item does not exist"
assert inventory.remove_item("Apple", 20) == "Item removed successfully"
assert inventory.get_stock("Apple") == 30
assert inventory.remove_item("Apple", 40) == "Not enough stock to remove"
assert inventory.remove_item("Orange", 10) == "Item does not exist"
print("All test cases passed successfully!")
```

Output

```
PS C:\Users\ANIL\Desktop\AI Assisant> & C:/Users/ANIL/AppData/Local/Python/pythoncore-3.14-64/python.exe "c:/Users/ANIL/Desktop/AI Assisant/inventory
All test cases passed successfully!
PS C:\Users\ANIL\Desktop\AI Assisant>
```

Analysis:

1. AI-generated assert tests validated real-world scenarios like:
   - Adding stock
   - Removing stock
   - Checking non-existent items
   - Preventing over-removal
2. Since no assertion errors occurred, the system works correctly.

Task Description #5:
(Date Validation & Formatting – Apply AI for

Data Validation)

• Task: Use AI to generate at least 3 assert test cases for validate_and_format_date(date_str) to check and convert dates.

• Requirements:

o Validate "MM/DD/YYYY" format. o Handle invalid dates.

o Convert valid dates to "YYYY-MM-DD".

Example Assert Test Cases:

assert validate_and_format_date("10/15/2023") == "2023-10-15"

assert validate_and_format_date("02/30/2023") == "Invalid Date"

assert validate_and_format_date("01/01/2024") == "2024-01-01"

Expected Output #5:

• Function passes all AI-generated assertions and handles edge cases.

Prompt :

#Validates the date in "MM/DD/YYYY" format
#Handles invalid dates (e.g., February 30)
#Converts valid dates into "YYYY-MM-DD" format
#Generates at least 3 assert test cases including edge cases
Assert Test Cases from user input

Code :

```python
#Validates the date in "MM/DD/YYYY" format
#Handles invalid dates (e.g., February 30)
#Converts valid dates into "YYYY-MM-DD" format
#Generates at least 3 assert test cases including edge cases
from datetime import datetime
def validate_and_format_date(date_str):
    try:
        # Validate the date format and check if it's a valid date
        date_obj = datetime.strptime(date_str, "%m/%d/%Y")
        # Convert to "YYYY-MM-DD" format
        return date_obj.strftime("%Y-%m-%d")
    except ValueError:
        return "Invalid date"
# Assert Test Cases from user input
assert validate_and_format_date("02/28/2024") == "2024-02-28"  # Valid date
assert validate_and_format_date("02/30/2024") == "Invalid date"  # Invalid date
assert validate_and_format_date("13/01/2024") == "Invalid date"  # Invalid month
assert validate_and_format_date("12/31/2024") == "2024-12-31"  # Valid date
assert validate_and_format_date("00/10/2024") == "Invalid date"  # Invalid month
assert validate_and_format_date("01/01/2024") == "2024-01-01"  # Valid date
print("All test cases passed successfully!")
```

Output :

```
PS C:\Users\ANIL\Desktop\AI Assisant> & C:/Users/ANIL/AppData/Local/Python/pythoncore-3.14-64/python.exe "c:
All test cases passed successfully!
PS C:\Users\ANIL\Desktop\AI Assisant>
```

Analysis:

1. The function validates dates using Python's datetime module.
2. AI-generated assert cases helped verify correct and incorrect inputs.
3. Since no assertion errors occurred, the function works correctly.