A.Srinidhi

2303A51131

AI-8.1

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.

CODE:

```
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
user_password = input("Enter your password: ")
if is_strong_password(user_password):
    print("Strong Password ")
else:
    print("Weak Password ")
```

Steps / Logic:

1. **Check length:** Password must be at least 8 characters.
    - if len(password) < 8: return False
2. **Check for spaces:** Spaces are not allowed.
    - if " " in password: return False
3. **Check character types:** Password must include:
    - **Uppercase letters** (A-Z) → any(char.isupper() for char in password)
    - **Lowercase letters** (a-z) → any(char.islower() for char in password)
    - **Digits** (0-9) → any(char.isdigit() for char in password)
    - **Special characters** (!@#$%^&*()_+ etc.) → any(char in string.punctuation for char in password)
4. **Return result:** Only True if all conditions are satisfied

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.

CODE:

```
def classify_number(n):
    if n is None or not isinstance(n, (int, float)):
        return "Invalid Input"
    for _ in range(1):
        if n > 0:
            return "Positive"
        elif n < 0:
            return "Negative"
        else:
            return "Zero"
assert classify_number(10) == "Positive"
assert classify_number(-5) == "Negative"
assert classify_number(0) == "Zero"
assert classify_number(1) == "Positive"
assert classify_number(-1) == "Negative"
assert classify_number("123") == "Invalid Input"
assert classify_number(None) == "Invalid Input"
user_input = input("Enter a number: ")
try:
    number = float(user_input)
except:
    number = None
result = classify_number(number)
print("Classification:", result)
```

Steps / Logic:

1. **Check input type:** Only allow integers or floats.
    - Invalid inputs (string, None, list) → "Invalid Input"
2. **Use a loop** (requirement): Can use a single iteration loop to demonstrate.
    - for _ in range(1): …
3. **Classify number:**
    - n > 0 → "Positive"
    - n < 0 → "Negative"
    - n == 0 → "Zero"

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.

CODE:

```
import string
def is_anagram(str1, str2):
   if str1 is None or str2 is None:
      return False
   cleaned1 = ""
   cleaned2 = ""
```

```
    for char in str1.lower():
        if char.isalnum():
            cleaned1 += char
    for char in str2.lower():
        if char.isalnum():
            cleaned2 += char
    if cleaned1 == "" and cleaned2 == "":
        return True
    return sorted(cleaned1) == sorted(cleaned2)
first_string = input("Enter first string: ")
second_string = input("Enter second string: ")
if is_anagram(first_string, second_string):
    print("Result: The strings are Anagrams ")
else:
    print("Result: The strings are NOT Anagrams ")
```

Steps / Logic:

1. **Normalize strings:**
    - Convert to lowercase: str.lower()
    - Remove spaces and punctuation: char.isalnum()
2. **Edge cases:**
    - Empty strings → consider as anagrams.
    - None → not anagrams.
3. **Compare sorted letters:**
    - If sorted(cleaned_str1) == sorted(cleaned_str2) → True
    - Else → False

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.

CODE:

```
from datetime import datetime
def validate_and_format_date(date_str):
    try:
        dt = datetime.strptime(date_str, "%m/%d/%Y")
        return dt.strftime("%Y-%m-%d")
    except ValueError:
        return "Invalid Date"
print("=== Date Validator & Formatter ===")
while True:
    user_input = input("Enter a date in MM/DD/YYYY format (or type 'exit' to quit): ")
    if user_input.lower() == "exit":
        print("Exiting Date Validator. Goodbye!")
        break
    result = validate_and_format_date(user_input)
    if result == "Invalid Date":
        print("Error: Invalid Date ")
    else:
```

```
print(f"Formatted Date: {result} ")
```

Steps / Logic:

1. **Parse date using datetime.strptime()**
    - If invalid → raises ValueError → return "Invalid Date"
2. **Format date using strftime()**
    - Convert to "YYYY-MM-DD"
3. **User input loop:**
    - Ask user to enter date
    - Print formatted date or error message
    - Type "exit" to quit