AI ASSISTED CODING
ASSIGNMENT 8.1

M.KEERTHANA
2403A51259
BATCH-11
CSE-GEN

Task 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:

Task: Implement a password strength validator function

Requirements:

 -Password must have at least 8 characters

- Must include uppercase, lowercase, digit, and special character

- Must not contain spaces

Also generate at least 3 assert test cases for is_strong_password(password).

Expected Output:

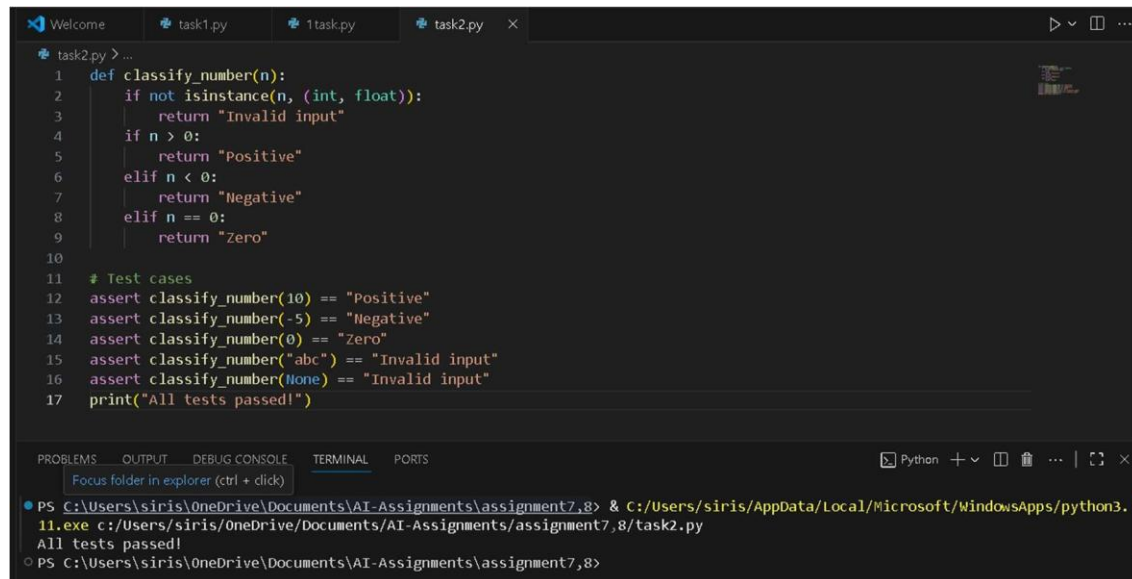Password validation logic passing all AI-generated test cases.

Code and Output:

```
Welcome          task1.py        1task.py  X

 1task.py > ...
  1    import re
  2
  3    def is_strong_password(password):
  4        # Check length
  5        if len(password) < 8:
  6            return False
  7        # Check for spaces
  8        if ' ' in password:
  9            return False
 10        # Check for uppercase
 11        if not re.search(r'[A-Z]', password):
 12            return False
 13        # Check for lowercase
 14        if not re.search(r'[a-z]', password):
 15            return False
 16        # Check for digit
 17        if not re.search(r'\d', password):
 18            return False
 19        # Check for special character
 20        if not re.search(r'[^A-Za-z0-9]', password):
 21            return False
 22        return True
 23
 24    # Test cases
 25    assert is_strong_password("Abcdef1!") == True
 26    assert is_strong_password("abcdef1!") == False  # No uppercase
 27    assert is_strong_password("Abcdefgh") == False  # No digit or special char
 28    assert is_strong_password("Abcdef1! ") == False # Contains space
 29    print("All tests passed!")

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\siris\OneDrive\Documents\AI-Assignments\assignment7,8> & C:/Users/siris/AppData/Local/MicrosoftPS C:\Users\siris\One
PS C:\Users\siris\OneDrive\Documents\AI-Assignments\assignment7,8> & C:/Users/siris/AppData/Local/Microsoft/WindowsApps/python3.
All tests passed!
PS C:\Users\siris\OneDrive\Documents\AI-Assignments\assignment7,8>
```

Task 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:

Task: Implement a Number Classification function with Loops
Requirements:

- Classify numbers as Positive, Negative, or Zero. -
Handle invalid inputs like strings and None.

Also generate at least 3 assert test cases for classify_number(n) Expected

Output:

Password validation logic passing all AI-generated test cases.

Code and output:

Task 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:

Task: Implement an Anagram Checker function is_anagram(str1, str2)

Requirements:

- Ignore case, spaces, and punctuation

- Handle edge cases (empty strings, identical words) Generate at

  least 3 assert test cases.

 Expected Output: Function correctly identifying anagrams and passing all AI-generated tests.

Code and output:

```
import re

def is_anagram(str1, str2):
    # Remove spaces and punctuation, convert to lowercase
    clean1 = re.sub(r'[\W_]', '', str1).lower()
    clean2 = re.sub(r'[\W_]', '', str2).lower()
    # Edge case: both empty strings are anagrams
    if clean1 == "" and clean2 == "":
        return True
    # Compare sorted characters
    return sorted(clean1) == sorted(clean2)

# Test cases
assert is_anagram("Listen", "Silent") == True
assert is_anagram("Astronomer", "Moon starer") == True
assert is_anagram("Hello", "Olelh!") == True
assert is_anagram("Test", "Taste") == False
assert is_anagram("", "") == True
print("All tests passed!")
```

```
PS C:\Users\siris\OneDrive\Documents\AI-Assignments\assignment7,8> & C:/Users/siris/AppData/Local/Microsoft/WindowsApps/python3.
11.exe c:/Users/siris/OneDrive/Documents/AI-Assignments/assignment7,8/task3.py
All tests passed!
PS C:\Users\siris\OneDrive\Documents\AI-Assignments\assignment7,8>
```

Task 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:

Task: Implement an Inventory class with methods add_item, remove_item, get_stock.

Requirements:

- Manage stock quantities for items

- Generate at least 3 assert test cases

Expected Output: Fully functional class passing all assertions.

Code and output:

```python
class Inventory:
    def __init__(self):
        self.stock = {}

    def add_item(self, item, quantity):
        if quantity < 0:
            raise ValueError("Quantity cannot be negative")
        self.stock[item] = self.stock.get(item, 0) + quantity

    def remove_item(self, item, quantity):
        if item not in self.stock or self.stock[item] < quantity or quantity < 0:
            return False
        self.stock[item] -= quantity
        if self.stock[item] == 0:
            del self.stock[item]
        return True

    def get_stock(self, item):
        return self.stock.get(item, 0)

# Test cases
inv = Inventory()
inv.add_item("apple", 10)
assert inv.get_stock("apple") == 10

inv.add_item("banana", 5)
assert inv.get_stock("banana") == 5

assert inv.remove_item("apple", 3) == True
assert inv.get_stock("apple") == 7

assert inv.remove_item("banana", 10) == False  # Not enough stock
assert inv.get_stock("banana") == 5
```

PROBLEMS 1   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   ▶ Python + ∨ ▯ 🗑 … | 🗗 ×

```
/WindowsApps/python3.11.exe c:/Users/siris/OneDrive/Documents/AI-Assignments/assignment7,8/task4.py
PS C:\Users\siris\OneDrive\Documents\AI-Assignments\assignment7,8>
```

Task 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:

Task: Implement validate_and_format_date(date_str) Requirements:

- Validate "MM/DD/YYYY" format

- Handle invalid dates

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

Generate at least 3 assert test cases

Expected Output: Function passes all AI-generated assertions and handles edge cases.

Code and output:

```python
from datetime import datetime

def validate_and_format_date(date_str):
    try:
        # Try to parse the date in MM/DD/YYYY format
        dt = datetime.strptime(date_str, "%m/%d/%Y")
        # Return in YYYY-MM-DD format
        return dt.strftime("%Y-%m-%d")
    except ValueError:
        return "Invalid date"

# Test cases
assert validate_and_format_date("12/31/2022") == "2022-12-31"
assert validate_and_format_date("02/29/2021") == "Invalid date"  # 2021 is not a leap year
assert validate_and_format_date("13/01/2022") == "Invalid date"  # Invalid month
assert validate_and_format_date("04/31/2022") == "Invalid date"  # April has only 30 days
assert validate_and_format_date("01/01/2020") == "2020-01-01"
```

```
PS C:\Users\siris\OneDrive\Documents\AI-Assignments\assignment7,8> & C:/Users/siris/AppData/Local/Microsoft/WindowsApps/python3.
11.exe c:/Users/siris/OneDrive/Documents/AI-Assignments/assignment7,8/task5.py
PS C:\Users\siris\OneDrive\Documents\AI-Assignments\assignment7,8>
```