ASSIGNMENT_8.1

HT.NO:2303A51731

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:

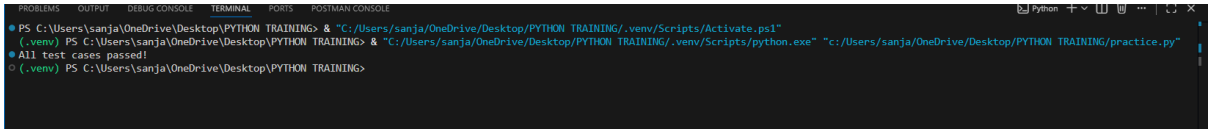
• Password validation logic passing all AI-generated test cases.

CODE:

```python
#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
• Password validation logic passing all AI-generated test cases"""
#fix the code below

import re
def is_strong_password(password):
    if len(password) < 8:
        return False
    if ' ' in password:
        return False
    if not re.search(r'[A-Z]', password):
        return False
    if not re.search(r'[a-z]', password):
        return False
    if not re.search(r'[0-9]', password):
        return False
    if not re.search(r'[\W_]', password):
        return False
    return True
# AI-generated 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("A1b2C3d4!") == True
assert is_strong_password("Short1!") == False
assert is_strong_password("NoSpecialChar1") == False
assert is_strong_password("With Space1@") == False
assert is_strong_password("ValidPass#9") == True
assert is_strong_password("nouppercase1@") == False
assert is_strong_password("NOLOWERCASE1@") == False
assert is_strong_password("NoDigit@") == False
print("All test cases passed!")
```

OUTPUT:



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.

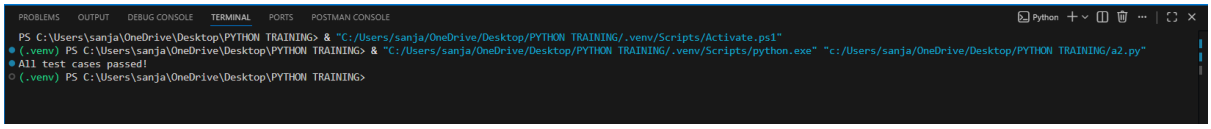
CODE:

practice.py a2.py

a2.py > ...

```
#Task Description #2 (Number Classification with Loops – ApplyAI 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."""
def classify_number(n):
    if not isinstance(n, (int, float)):
        return "Invalid input"
    if n > 0:
        return "Positive"
    elif n < 0:
        return "Negative"
    else:
        return "Zero"
# Assert Test Cases
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("string") == "Invalid input"
assert classify_number(-0.0001) == "Negative"
print("All test cases passed!")
```

OUTPUT:



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:

practice.py | a2.py

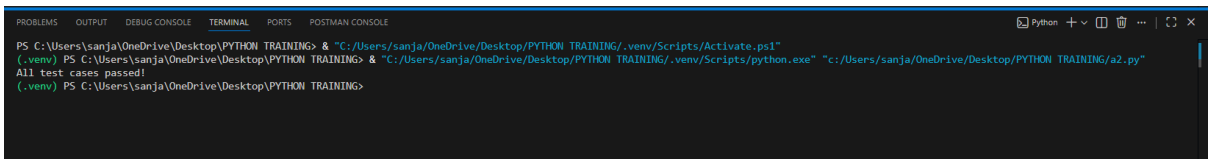
C:\Users\sanja\OneDrive\Desktop\PYTHON TRAINING\practice.py

```python
#Task Description #3 (Anagram Checker – Apply AI for StringAnalysis)
# 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."""


def is_anagram(str1, str2):
    # Remove non-alphabetic characters and convert to lowercase
    cleaned1 = ''.join(c.lower() for c in str1 if c.isalpha())
    cleaned2 = ''.join(c.lower() for c in str2 if c.isalpha())
    # Check if sorted characters are equal
    return sorted(cleaned1) == sorted(cleaned2)

# Assert Test Cases
assert is_anagram("listen", "silent") == True
assert is_anagram("hello", "world") == False
assert is_anagram("Dormitory", "Dirty Room") == True
assert is_anagram("Astronomer", "Moon starer") == True
assert is_anagram("The Morse Code", "Here come dots!") == True
assert is_anagram("", "") == True  # Edge case: both strings empty
assert is_anagram("a", "A") == True  # Edge case: single character, different cases
assert is_anagram("abc", "ab") == False  # Edge case: different lengths
assert is_anagram("123", "321") == True  # Edge case: numeric characters ignored
assert is_anagram("A gentleman", "Elegant man") == True  # Edge case: spaces and cases ignored
print("All test cases passed!")
```

OUTPUT:



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.

CODE:

```python
#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:
"""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"""
#use assert Ask AI to generate at least 3 assert-based tests for an Inventory class with stock management.
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 in self.stock and self.stock[name] >= quantity:
            self.stock[name] -= quantity
        else:
            raise ValueError("Not enough stock to remove")

    def get_stock(self, name):
        return self.stock.get(name, 0)
# 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
# Additional Assert Test Cases:
inv.add_item("Notebook", 7)
assert inv.get_stock("Notebook") == 7
inv.remove_item("Notebook", 2)
assert inv.get_stock("Notebook") == 5
inv.add_item("Eraser", 4)
assert inv.get_stock("Eraser") == 4
inv.remove_item("Eraser", 4)
assert inv.get_stock("Eraser") == 0
print("All tests passed!")
```

OUTPUT:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  POSTMAN CONSOLE

PS C:\Users\sanja\OneDrive\Desktop\PYTHON TRAINING> & "C:/Users/sanja/OneDrive/Desktop/PYTHON TRAINING/.venv/Scripts/Activate.ps1"
(.venv) PS C:\Users\sanja\OneDrive\Desktop\PYTHON TRAINING> & "C:/Users/sanja/OneDrive/Desktop/PYTHON TRAINING/.venv/Scripts/python.exe" "c:/Users/sanja/OneDrive/Desktop/PYTHON TRAINING/a2.py"
All tests passed!
(.venv) PS C:\Users\sanja\OneDrive\Desktop\PYTHON TRAINING>
```

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:

```python
#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.
"""
from datetime import datetime
def validate_and_format_date(date_str):
    try:
        # Parse the date string to a datetime object
        date_obj = datetime.strptime(date_str, "%m/%d/%Y")
        # Return the date in "YYYY-MM-DD" format
        return date_obj.strftime("%Y-%m-%d")
    except ValueError:
        # If parsing fails, return "Invalid Date"
        return "Invalid Date"
# AI-generated 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"


print("All test cases passed!")
```

OUTPUT:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  POSTMAN CONSOLE

PS C:\Users\sanja\OneDrive\Desktop\PYTHON TRAINING> & "C:/Users/sanja/OneDrive/Desktop/PYTHON TRAINING/.venv/Scripts/Activate.ps1"
(.venv) PS C:\Users\sanja\OneDrive\Desktop\PYTHON TRAINING> & "C:/Users/sanja/OneDrive/Desktop/PYTHON TRAINING/.venv/Scripts/python.exe" "c:/Users/sanja/OneDrive/Desktop/PYTHON TRAINING/a
All test cases passed!
(.venv) PS C:\Users\sanja\OneDrive\Desktop\PYTHON TRAINING>
```