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:

- Password must have at least 8 characters.
- Must include uppercase, lowercase, digit, and special character.
- 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 :

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

Prompt:

#Develop a python code that geneartes atleast 3 assert test cases for is_strong_password(passwoed) and implement the validator function.

#criteria for a strong password:

#1. At least 8 characters long

#2. Contains at least one uppercase letter

#3. Contains at least one lowercase letter

#4. Contains at least one digit

#5. Contains at least one special character (e.g., !@#$%^&*)

Code:

```
import re

def is_strong_password(password):

    if len(password) < 8:

        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'[!@#$%^&*]', password):

        return False

    return True

# Test cases
```

assert is_strong_password("StrongPass1!") == True, "Test case 1 failed"

assert is_strong_password("weakpass") == False, "Test case 2 failed"

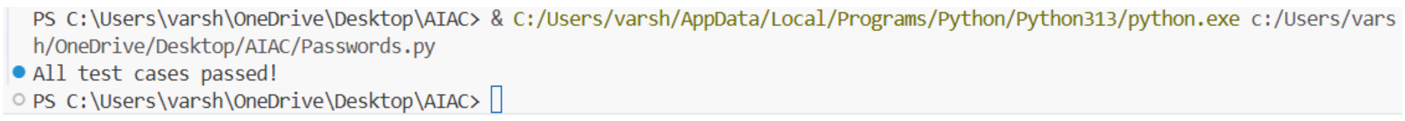
assert is_strong_password("Short1!") == False, "Test case 3 failed"

assert is_strong_password("NoSpecialChar1") == False, "Test case 4 failed"

assert is_strong_password("NoDigit!@#") == False, "Test case 5 failed"

print("All test cases passed!")

Output Screenshot:



```
PS C:\Users\varsh\OneDrive\Desktop\AIAC> & C:/Users/varsh/AppData/Local/Programs/Python/Python313/python.exe c:/Users/vars
h/OneDrive/Desktop/AIAC/Passwords.py
● All test cases passed!
○ PS C:\Users\varsh\OneDrive\Desktop\AIAC> 
```

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:

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

• Classification logic passing all assert tests.

Prompt:

#Develop a python program whichgenerate at least 3 assert test cases for a classify_number(n) function. Implement using loops.

#The classify_number(n) function should classify a given number n as "Positive", "Negative", or "Zero". The program should handle edge cases and ensure that the input is a valid number.

Code:

```
def classify_number(n):

    if n > 0:

        return "Positive"

    elif n < 0:

        return "Negative"

    else:

        return "Zero"

# Test cases
```

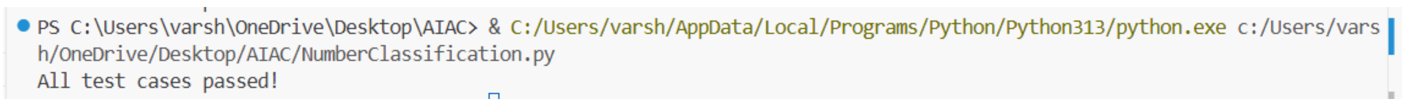
assert classify_number(10) == "Positive", "Test case 1 failed"

assert classify_number(-5) == "Negative", "Test case 2 failed"

assert classify_number(0) == "Zero", "Test case 3 failed"

print("All test cases passed!")

Output Screenshot:



```
PS C:\Users\varsh\OneDrive\Desktop\AIAC> & C:/Users/varsh/AppData/Local/Programs/Python/Python313/python.exe c:/Users/vars
h/OneDrive/Desktop/AIAC/NumberClassification.py
All test cases passed!
```

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:

- Ignore case, spaces, and punctuation.
- 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:

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

Prompt:

#Write a Python function is_anagram(str1, str2) that returns True if both strings are anagrams of each other (ignoring spaces and case), otherwise returns False

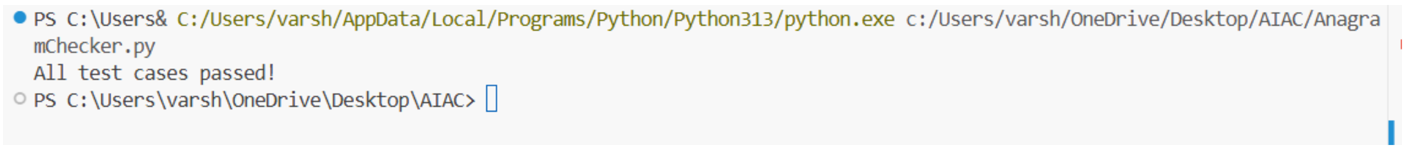
Code:

```
def is_anagram(str1, str2):
    # Remove spaces and convert to lowercase
    str1 = str1.replace(" ", "").lower()
    str2 = str2.replace(" ", "").lower()

    # Sort the characters of both strings and compare
    return sorted(str1) == sorted(str2)
# Test cases
assert is_anagram("listen", "silent") == True, "Test case 1 failed"
assert is_anagram("triangle", "integral") == True, "Test case 2 failed"
assert is_anagram("apple", "pabble") == False, "Test case 3 failed"
assert is_anagram("Dormitory", "Dirty Room") == True, "Test case 4 failed"
```

assert is_anagram("Conversation", "Voices Rant On") == True, "Test case 5 failed"

print("All test cases passed!")

Output Screenshot:



```
PS C:\Users& C:/Users/varsh/AppData/Local/Programs/Python/Python313/python.exe c:/Users/varsh/OneDrive/Desktop/AIAC/AnagramChecker.py
All test cases passed!
PS C:\Users\varsh\OneDrive\Desktop\AIAC>
```

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:

- add_item(name, quantity)
- remove_item(name, quantity)
- 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:

• Fully functional class passing all assertions.

Prompt:

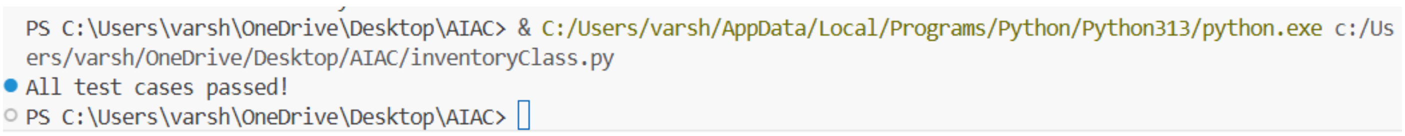
#Create a class Inventory with methods to manage items: add_item(name, quantity), remove_item(name, quantity), get_stock(name) using atleast 3 assert test cases for each method. The program should handle edge cases such as trying to remove more items than available or checking stock for an item that doesn't exist.

Code:

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

  def add_item(self, name, quantity):
    if quantity < 0:
      return "Invalid input: Quantity cannot be negative."
    if name in self.items:
            self.items[name] += quantity
        else:
            self.items[name] = quantity

    def remove_item(self, name, quantity):
        if name not in self.items:
            return "Error: Item does not exist."
        if quantity < 0:
            return "Invalid input: Quantity cannot be negative."
        if self.items[name] < quantity:
            return "Error: Not enough stock to remove."
        self.items[name] -= quantity
        if self.items[name] == 0:
            del self.items[name]

    def get_stock(self, name):
        if name not in self.items:
            return "Error: Item does not exist."
        return self.items[name]
# Test cases
inventory = Inventory()
# Test add_item
inventory.add_item("Apple", 10)
assert inventory.get_stock("Apple") == 10, "Test case 1 failed"
inventory.add_item("Apple", 5)
assert inventory.get_stock("Apple") == 15, "Test case 2 failed"
assert inventory.add_item("Banana", -5) == "Invalid input: Quantity cannot be negative.", "Test case 3 failed"
# Test remove_item
assert inventory.remove_item("Apple", 5) == None, "Test case 4 failed"
assert inventory.get_stock("Apple") == 10, "Test case 5 failed"
assert inventory.remove_item("Apple", 15) == "Error: Not enough stock to remove.", "Test case 6 failed"
assert inventory.remove_item("Orange", 5) == "Error: Item does not exist.", "Test case 7 failed"
# Test get_stock
assert inventory.get_stock("Apple") == 10, "Test case 8 failed"
assert inventory.get_stock("Banana") == "Error: Item does not exist.", "Test case 9 failed"
```

print("All test cases passed!")

Output Screenshot:



```
PS C:\Users\varsh\OneDrive\Desktop\AIAC> & C:/Users/varsh/AppData/Local/Programs/Python/Python313/python.exe c:/Us
ers/varsh/OneDrive/Desktop/AIAC/inventoryClass.py
All test cases passed!
PS C:\Users\varsh\OneDrive\Desktop\AIAC>
```

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:

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

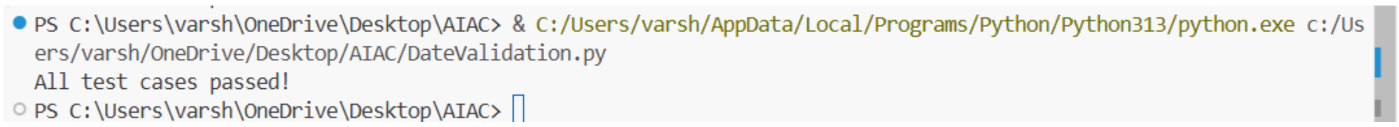
Prompt:

#Write a function validate_and_format_date(date_str) that: Validates if a date is in "MM/DD/YYYY" format.Returns "Invalid Date" for incorrect ones. Converts valid dates to "YYYY-MM-DD" using atleast 3 assert test cases for the function. The program should handle edge cases such as invalid month, day, or year values.

Code:

```
import re
def validate_and_format_date(date_str):
    # Check if the date is in the correct format using regex
    if not re.match(r'^\d{2}/\d{2}/\d{4}$', date_str):
        return "Invalid Date"

    month, day, year = map(int, date_str.split('/'))

    # Validate month
    if month < 1 or month > 12:
        return "Invalid Date"

    # Validate day based on month and leap year
    if day < 1:
        return "Invalid Date"
    if month in [1, 3, 5, 7, 8, 10, 12] and day > 31:
        return "Invalid Date"
    if month in [4, 6, 9, 11] and day > 30:
        return "Invalid Date"
    if month == 2:
        if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
            if day > 29:
                return "Invalid Date"
        else:
            if day > 28:
                return "Invalid Date"

    # If valid, convert to "YYYY-MM-DD"
    return f"{year:04d}-{month:02d}-{day:02d}"
# Test cases
assert validate_and_format_date("12/25/2020") == "2020-12-25", "Test case 1 failed"
assert validate_and_format_date("02/29/2020") == "2020-02-29", "Test case 2 failed"  # Leap year
assert validate_and_format_date("02/30/2020") == "Invalid Date", "Test case 3 failed"  # Invalid day
assert validate_and_format_date("13/01/2020") == "Invalid Date", "Test case 4 failed"  # Invalid month
assert validate_and_format_date("00/10/2020") == "Invalid Date", "Test case 5 failed"  # Invalid month
assert validate_and_format_date("01/00/2020") == "Invalid Date", "Test case 6 failed"  # Invalid day
assert validate_and_format_date("01/32/2020") == "Invalid Date", "Test case 7 failed"  # Invalid day
assert validate_and_format_date("02/29/2019") == "Invalid Date", "Test case 8 failed"  # Not a leap year
assert validate_and_format_date("invalid_date") == "Invalid Date", "Test case 9 failed"  # Invalid format
print("All test cases passed!")
```

Output Screenshot:



```
PS C:\Users\varsh\OneDrive\Desktop\AIAC> & C:/Users/varsh/AppData/Local/Programs/Python/Python313/python.exe c:/Us
ers/varsh/OneDrive/Desktop/AIAC/DateValidation.py
All test cases passed!
PS C:\Users\varsh\OneDrive\Desktop\AIAC>
```