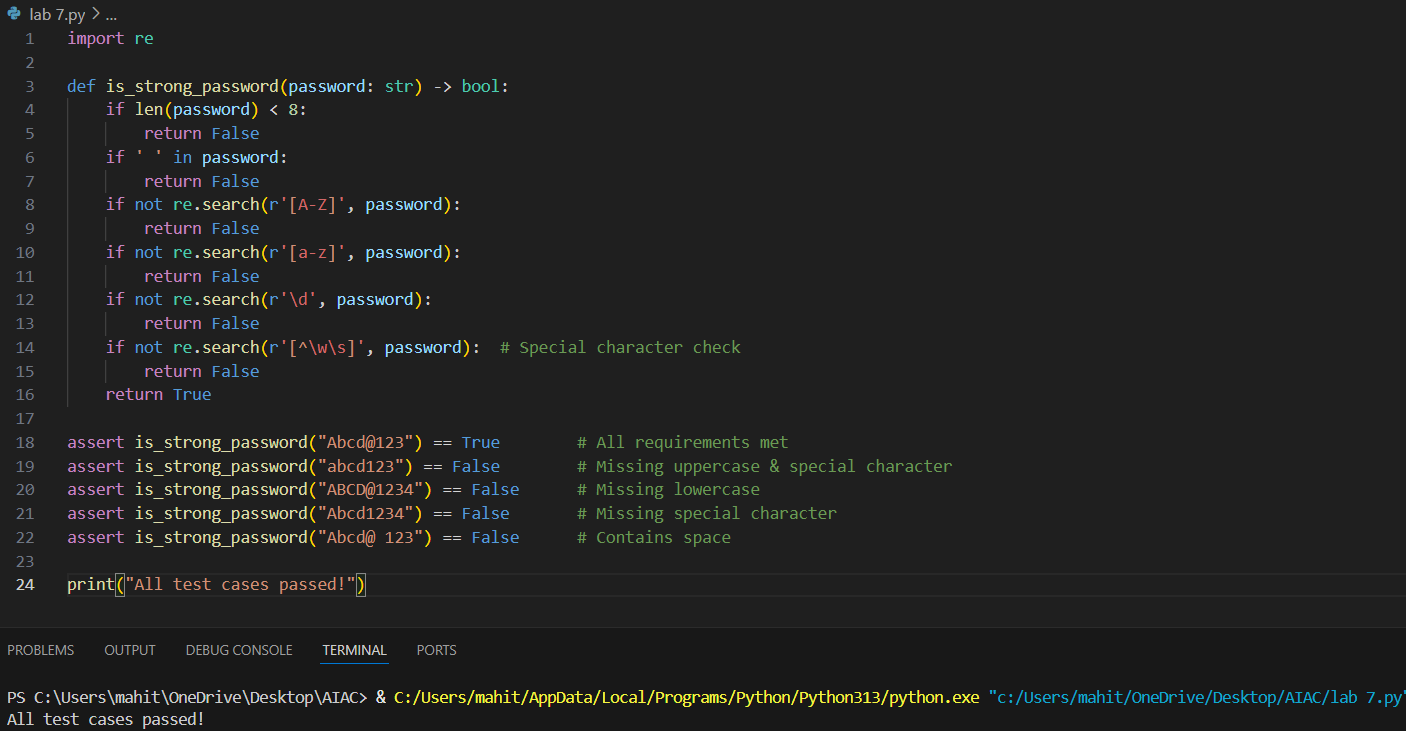
# LAB ASSIGNMENT-8

TASK-1:

Prompt:

Build a password strength validator, which is a critical part of application security. Weak passwords are one of the biggest security risks. Generate at least 3 assert test cases for is\_strong\_password(password) and implement the validator function.  
Example Assert Test Cases: assert is\_strong\_password("Abcd@123") == True assert is\_strong\_password("abcd123") == False assert is\_strong\_password("ABCD@1234") == True Requirements: Password must have at least 8 characters. Must include uppercase, lowercase, digit, and special character. Must not contain spaces.

Code and Output:



Code Explanation:

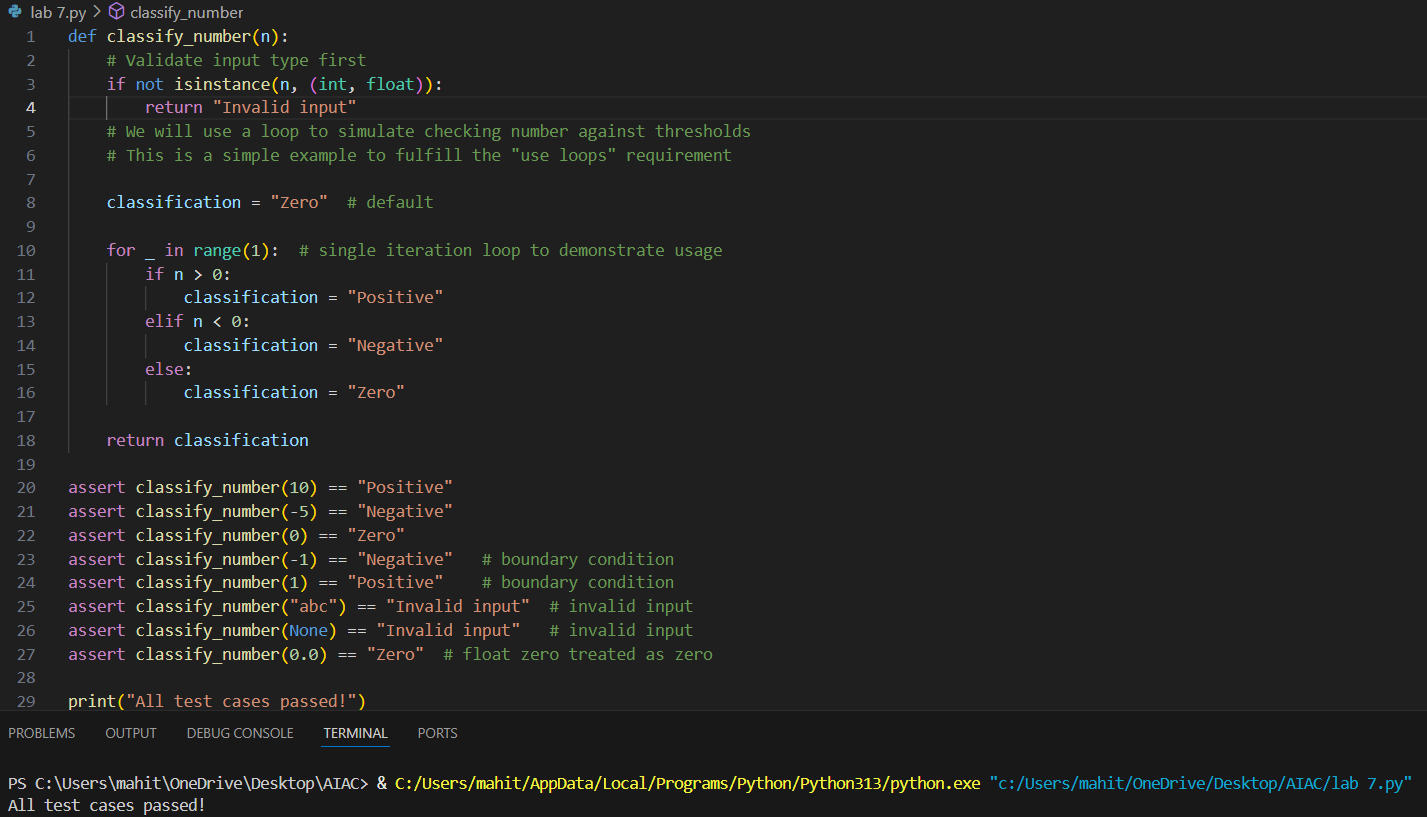
* The function is\_strong\_password(password) checks if a password meets strong security standards.
* It uses the re module (regular expressions) to search for specific character types.
* The password must be **at least 8 characters** long and **must not contain spaces**.
* It must include at least **one uppercase letter**, **one lowercase letter**, **one digit**, and **one special character** (like @, #, etc.).
* If any of these checks fail, the function returns False; otherwise, it returns True.
* Several assert statements are used to **test different password cases** to verify the function works correctly.
* If all assertions pass, it prints: "All test cases passed!".
* This helps ensure only strong and secure passwords are accepted

TASK-2:

Prompt:

Build a Number Classification with Loops, which is a Edge Case Handling. Generate at least 3 assert test cases for a classify\_number(n) function. Implement using loops. Example Assert Test Cases: assert classify\_number(10) == "Positive" assert classify\_number(-5) == "Negative" assert classify\_number(0) == "Zero" • Requirements: o Classify numbers as Positive, Negative, or Zero. o Handle invalid inputs like strings and None. o Include boundary conditions (-1, 0, 1).

Code and Output:



Code Explanation:

* The function first checks if the input n is a number (int or float). If not, it returns "Invalid input".
* It initializes a variable classification with "Zero" as the default value.
* A for loop with a single iteration (for \_ in range(1)) is used to demonstrate loop usage, even though it only runs once.

Inside the loop:

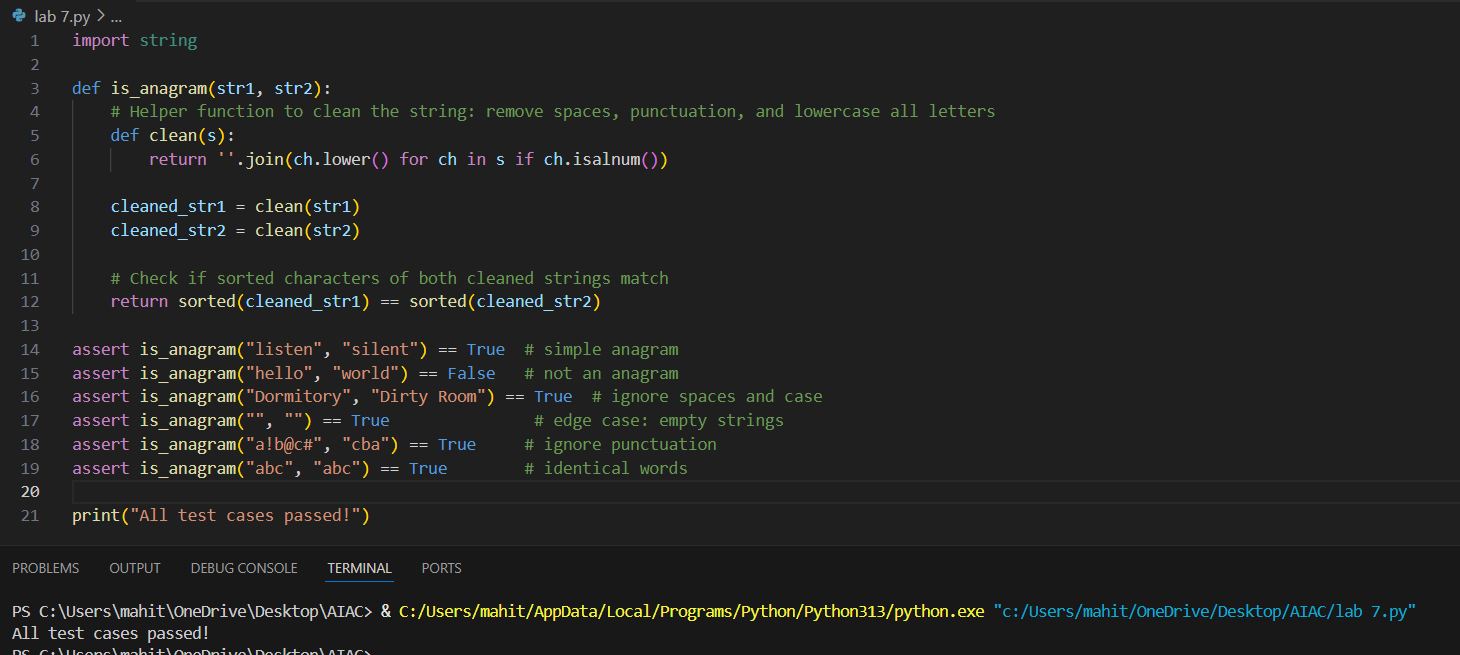
* If n is greater than 0, it sets classification to "Positive".
* If n is less than 0, it sets classification to "Negative".
* Otherwise, it remains "Zero".
* Finally, the function returns the classification string.
* This approach ensures edge cases (-1, 0, 1) and invalid inputs (None, strings) are properly handled.

TASK-3:

Prompt:

Anagram Checker - for String Analysis 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 Make sure that Function correctly identifying anagrams and passing all generated tests.

Code and Output:



Code Explanation:

* The function checks if two strings are anagrams, meaning they contain the same letters in any order.

It first defines a helper function clean(s) that:

* Converts the string to **lowercase** to ignore case differences.
* Removes all **spaces** and **punctuation** by keeping only alphanumeric characters (ch.isalnum()).
* Both input strings str1 and str2 are cleaned using this helper function.
* The cleaned strings are then **sorted alphabetically**.
* If the sorted versions of both cleaned strings are equal, the function returns True (they are anagrams).
* Otherwise, it returns False.
* This approach handles edge cases such as empty strings, identical words, and ignores spaces and punctuation, fulfilling the requirements.

TASK-4:

Prompt:

Inventory Class – Simulate Real-World Inventory System 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

Make sure functional class passing all assertions.

Code and Output:





Code Explanation:

* The Inventory class manages stock items using a dictionary called stock, where keys are item names and values are their quantities.
* **\_\_init\_\_ method:** Initializes an empty inventory dictionary.

**add\_item(name, quantity) method:**

* Adds the specified quantity of an item to the inventory.
* If the item doesn’t exist, it creates a new entry.
* It raises an error if a negative quantity is given.

**remove\_item(name, quantity) method**

* Removes the specified quantity from the given item.
* Raises errors if the item doesn’t exist, if quantity is negative, or if there isn’t enough stock.
* If after removal the stock reaches zero, the item is deleted from the inventory.

**get\_stock(name) method:**

* Returns the current stock quantity of the item.
* Returns 0 if the item doesn’t exist.
* This class simulates a real-world inventory system with basic stock management and error handling.

TASK-5:

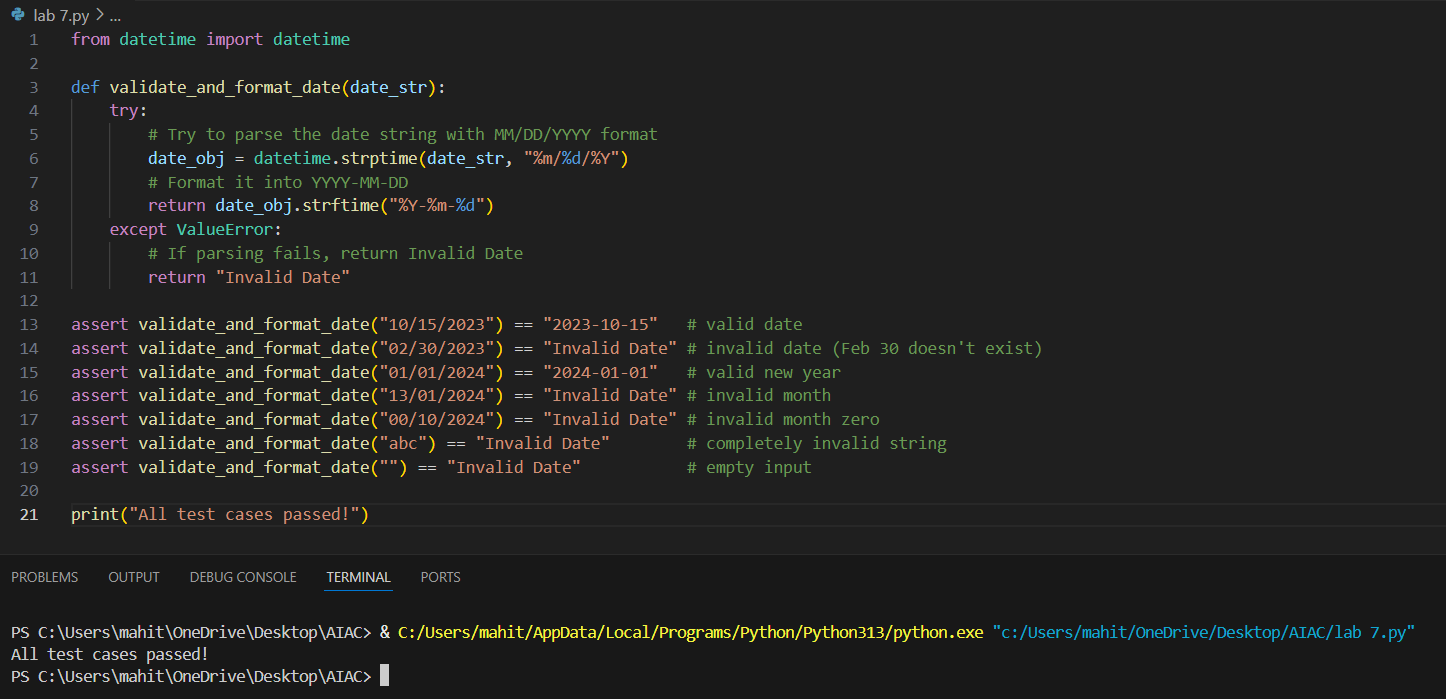
Prompt:

Date Validation & Formatting – for Data Validation Generate at least 3 assert test cases for validate\_and\_format\_date(date\_str) to check and convert dates.

• Requirements: 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" Make sure that Function passes all assertions and handles edge cases.

Code and Output:



Code Explanation:

* The function

validate\_and\_format\_date(date\_str) checks if a date is valid in "MM/DD/YYYY" format.

* It uses datetime.strptime() to try parsing the input string.
* If the date is valid, it's converted to "YYYY-MM-DD" format using strftime().
* If the input is invalid (wrong format or non-existent date), a ValueError is raised.
* The except block catches this error and returns "Invalid Date".
* This ensures the function handles both valid and invalid inputs safely.