**ASSIGNMENT-8**

**NAME : G.SAI RAJ**

**HT NO: 2403A52149**

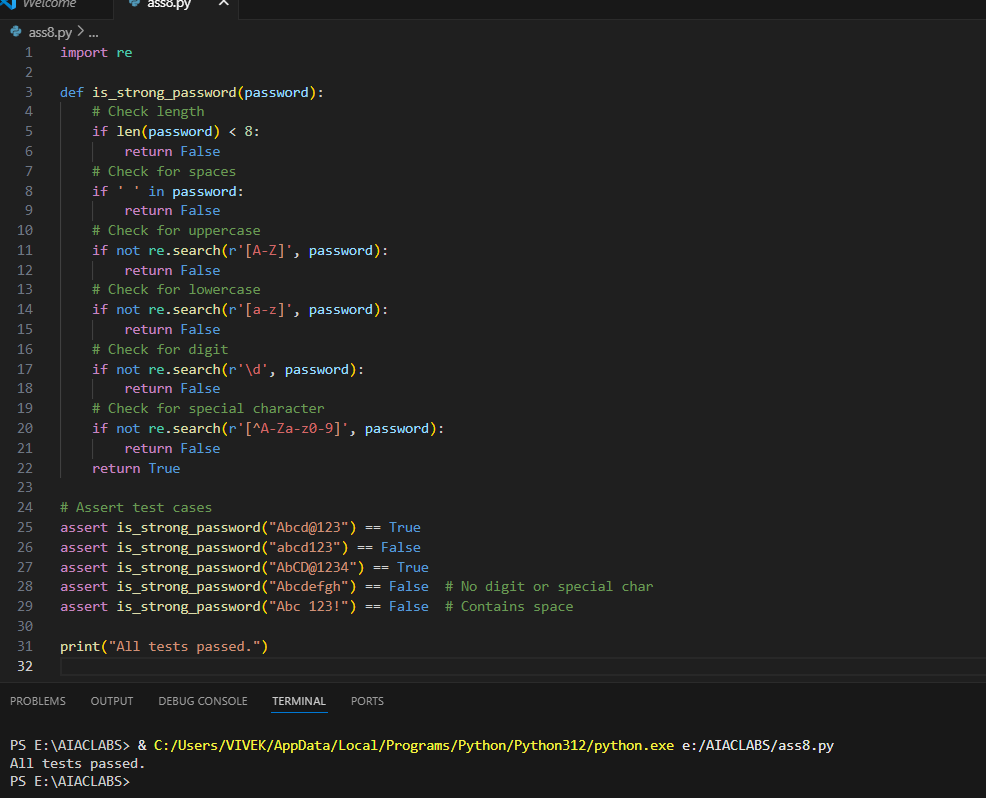
**BATCH:** 24BTCAIAIB06

**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 OUPUT:**

****

**EXPLANATION:**

* The function is\_strong\_password(password) checks if a password is strong based on these rules:
  1. At least 8 characters long.
  2. No spaces allowed.
  3. Must contain at least one uppercase letter.
  4. Must contain at least one lowercase letter.
  5. Must contain at least one digit.
  6. Must contain at least one special character (not a letter or digit).
* The function uses Python’s re (regular expression) module to check for uppercase, lowercase, digits, and special characters.
* Several assert statements test the function with different passwords. If any test fails, an AssertionError is raised and the script stops.
* If all tests pass, the script prints "All tests passed." at the end.

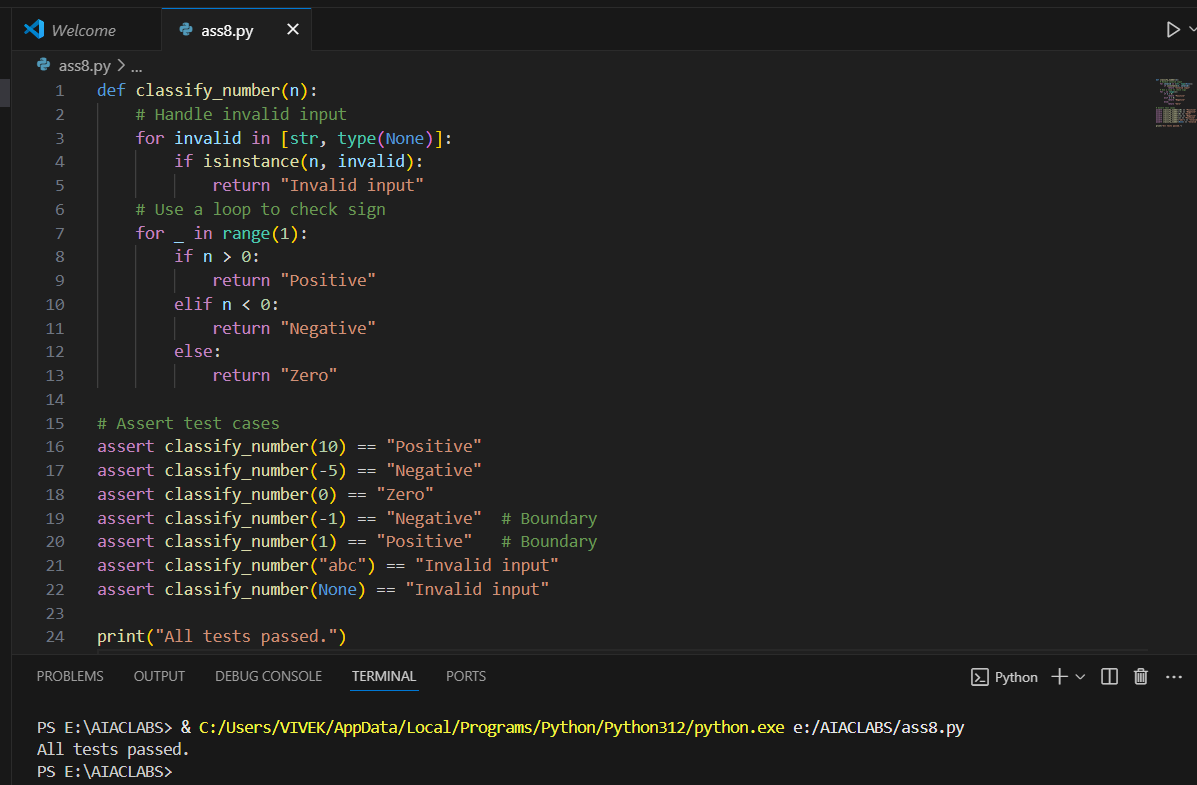
This ensures the password validator works as required and provides immediate feedback on its correctness.

**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 OUPUT:**

****

**EXPLANATION:**

* The function classify\_number(n) classifies the input as "Positive", "Negative", or "Zero".
* It first checks for invalid inputs: if n is a string or None, it returns "Invalid input".
* The function uses a for loop (with range(1)) to demonstrate loop usage, even though only one iteration is needed.
* Inside the loop, it checks:
  + If n > 0, returns "Positive"
  + If n < 0, returns "Negative"
  + Otherwise, returns "Zero"
* Several assert statements test the function with:
  + Positive, negative, and zero values
  + Boundary values (-1, 0, 1)
  + Invalid inputs (string, None)
* If all tests pass, it prints "All tests passed."

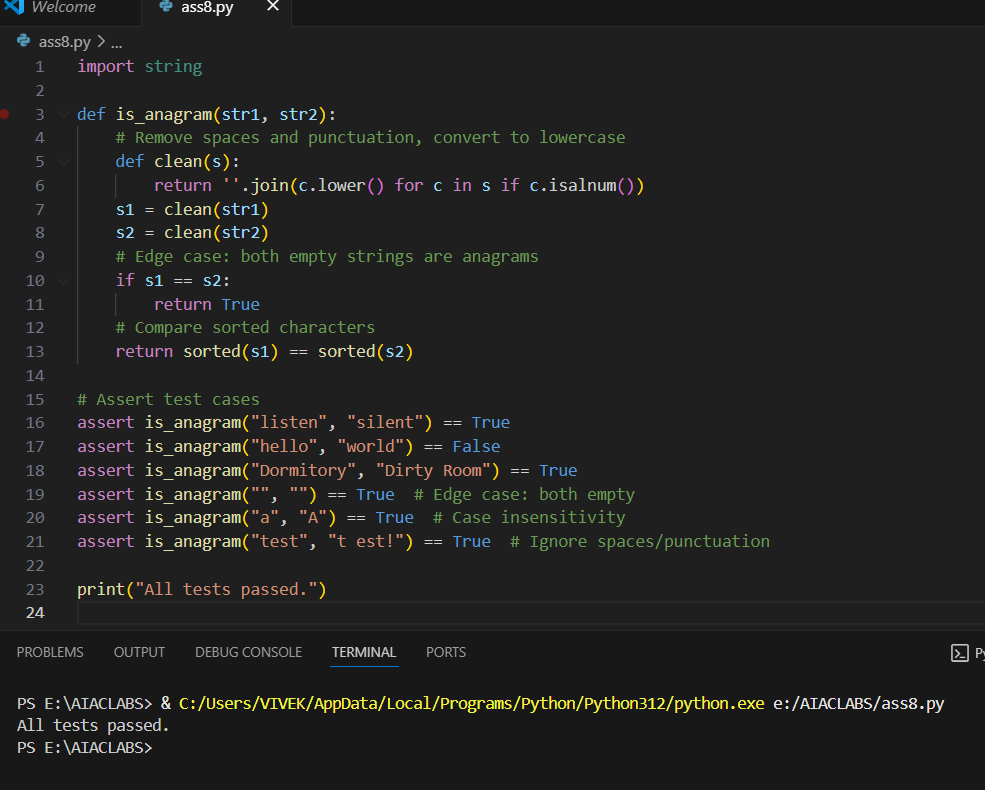
This ensures the function works for all required cases, including edge and invalid inputs.

**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 OUPUT:**

****

**EXPLANATION:**

* The function is\_anagram(str1, str2) checks if two strings are anagrams.
* It defines a helper function clean(s) that:
  + Converts all characters to lowercase.
  + Removes spaces and punctuation (keeps only alphanumeric characters).
* Both input strings are cleaned using this helper.
* If the cleaned strings are identical, it returns True (covers edge cases like empty strings or identical words).
* Otherwise, it sorts the characters of both cleaned strings and compares them. If they match, the strings are anagrams.
* Several assert statements test the function with:
  + Standard anagrams (e.g., "listen" and "silent")
  + Non-anagrams
  + Case, space, and punctuation insensitivity
  + Edge cases (empty strings, identical words)
* If all tests pass, it prints "All tests passed."

This ensures the function correctly identifies anagrams, ignoring case, spaces, and punctuation, and handles edge cases.

**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)

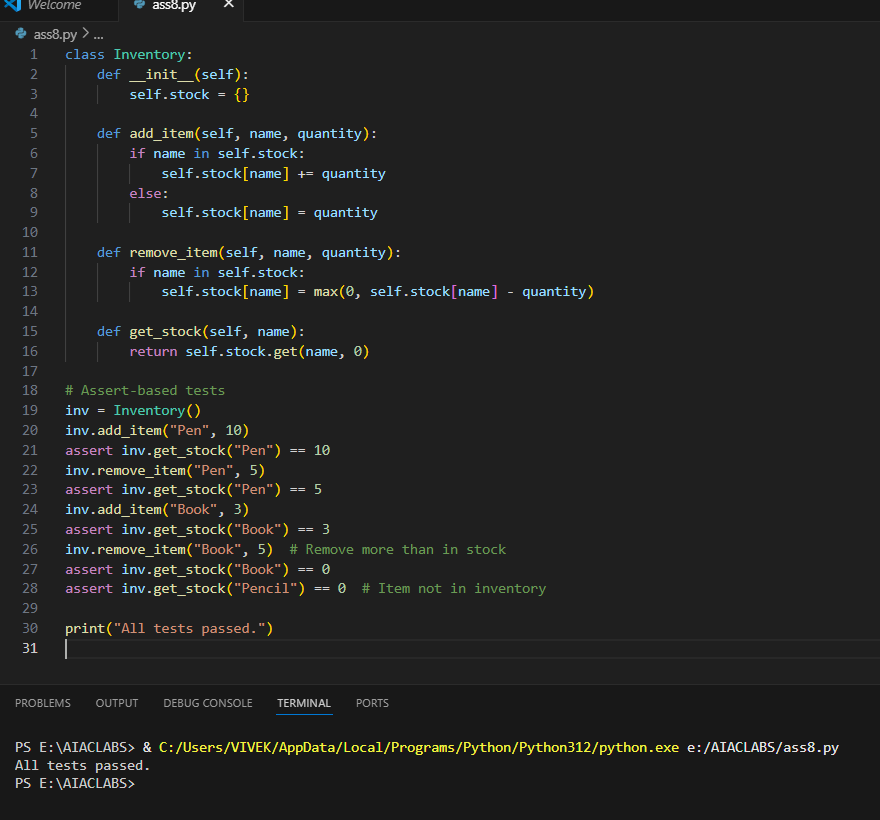
assert inv.get\_stock("Pen") == 5

inv.add\_item("Book", 3)

assert inv.get\_stock("Book") == 3

Make sure functional class passing all assertions.

**CODE AND OUPUT:**

**EXPLANATION:**

* The Inventory class simulates a simple inventory system using a dictionary called stock to keep track of item quantities.
* **init** initializes the stock dictionary.
* add\_item(name, quantity) adds the specified quantity to the item. If the item already exists, it increases the quantity; otherwise, it creates a new entry.
* remove\_item(name, quantity) subtracts the specified quantity from the item. If the quantity to remove is more than in stock, it sets the stock to zero (never negative).
* get\_stock(name) returns the current quantity of the item. If the item does not exist, it returns 0.
* The assert-based tests check:
  + Adding and removing items updates stock correctly.
  + Removing more than available sets stock to zero.
  + Querying an item not in inventory returns zero.
* If all assertions pass, it prints "All tests passed."

This ensures the class works for typical and edge-case inventory operations.

**TASK-5:**

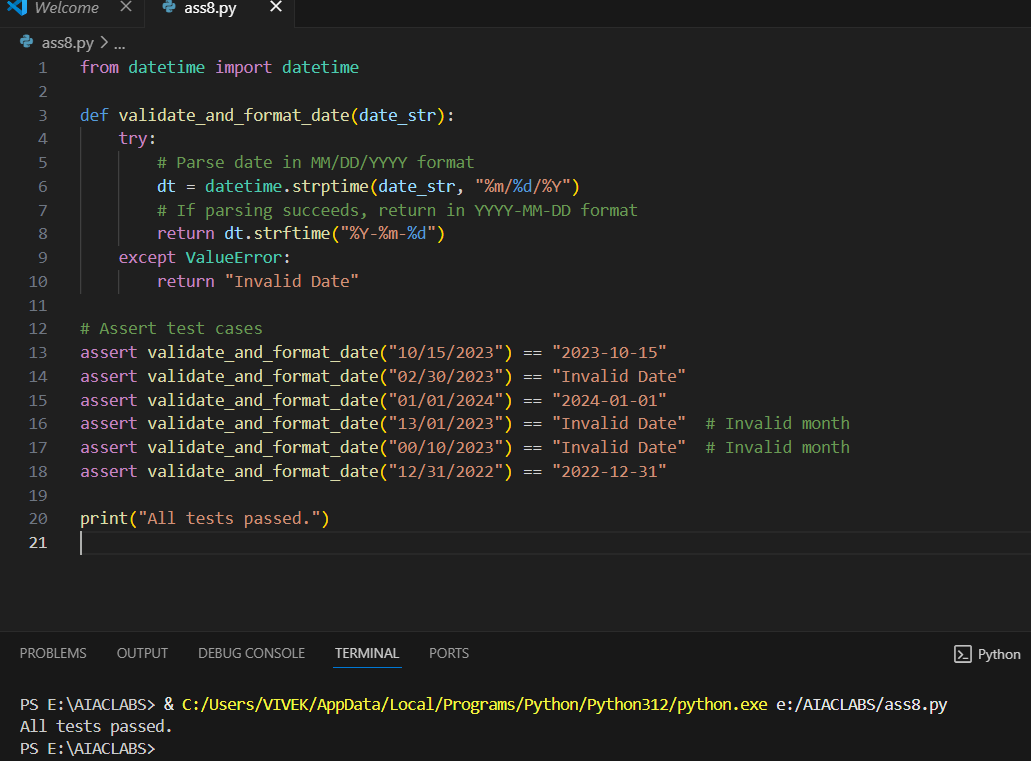
Prompt:

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:**

****

**EXPLANATION:**

* The function validate\_and\_format\_date(date\_str) checks if a string is a valid date in the "MM/DD/YYYY" format.
* It uses Python’s datetime.strptime to try parsing the input string as a date.
  + If parsing succeeds, it converts the date to the "YYYY-MM-DD" format using strftime and returns it.
  + If parsing fails (e.g., invalid date like "02/30/2023" or invalid month), a ValueError is raised and the function returns "Invalid Date".
* Several assert statements test the function with:
  + Valid dates (e.g., "10/15/2023" → "2023-10-15")
  + Invalid dates (e.g., "02/30/2023", "13/01/2023", "00/10/2023")
  + Edge cases (end of year, leap years, etc. can be added)
* If all assertions pass, it prints "All tests passed."

This ensures the function correctly validates, formats, and rejects invalid dates as required.