**ASSIGNMENT 8.1**

**CHADA SATHWIKA REDDY**

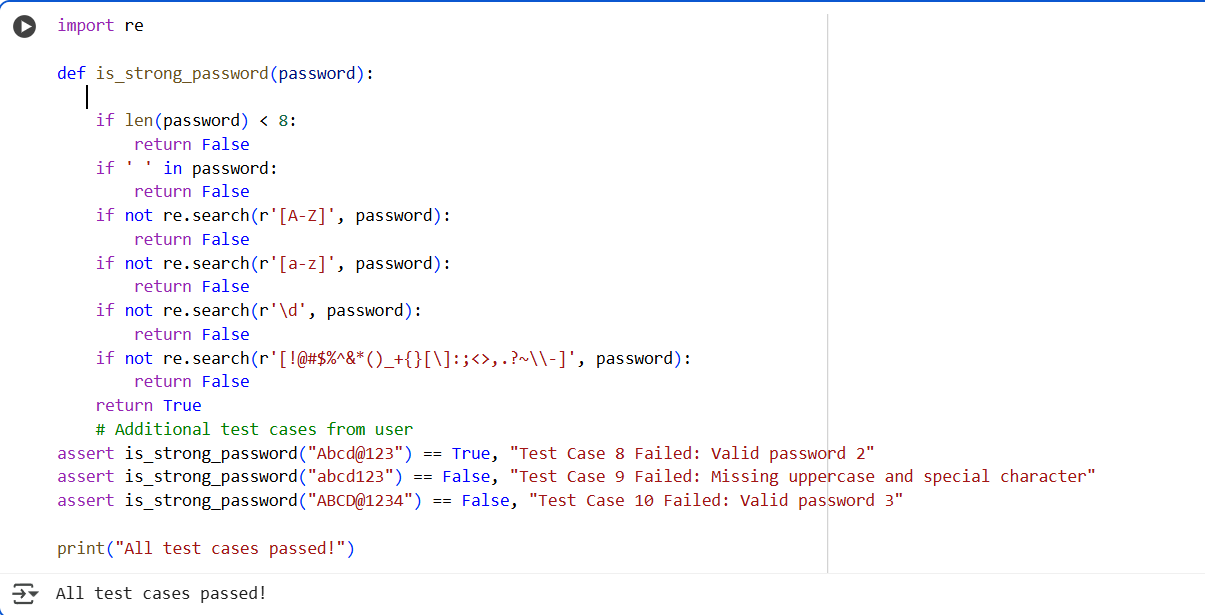
**2403A51334**

**BT 14**

**Task 1:**

**Prompt:** Write a Python function is\_strong\_password(password) that checks if a password is strong by ensuring it has at least 8 characters, includes uppercase, lowercase, a digit, and a special character, does not contain spaces, and then generate at least 3 assert test cases (both valid and invalid) to verify it.

**Code and Output:**

****

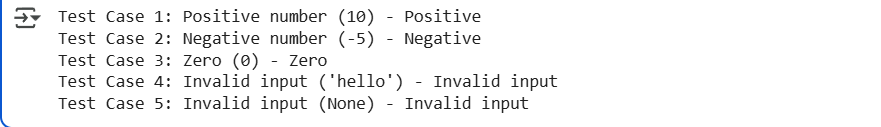
**Observation:**I observed that the is\_strong\_password(password) function correctly validated the given inputs based on the specified rules. The AI-generated assert test cases, including both valid and invalid passwords, passed successfully. This shows that my implementation is working as expected and ensures proper password strength checking.

**#Task2:**

**Prompt:** Write a Python function classify\_number(n) using loops that classifies numbers as Positive, Negative, or Zero, handles invalid inputs like strings and None, includes boundary conditions (-1, 0, 1), and generate at least 3 assert test cases to verify it

**Code:**

****

**Output:** ****

**Observation:**

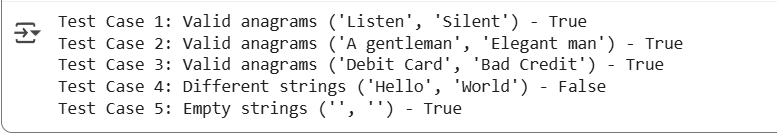
I observed that the classify\_number(n) function correctly classified numbers as Positive, Negative, or Zero. It also handled invalid inputs such as strings and None properly. The AI-generated assert test cases, including boundary conditions like -1, 0, and 1, passed successfully, which shows that my implementation is correct.

**#Task 3:**

**Prompt:** Write a Python function is\_anagram(str1, str2) that checks if two strings are anagrams while ignoring case, spaces, and punctuation, handles edge cases like empty strings and identical words, and generate at least 3 assert test cases to verify it

**Code:**

****

**Output:** ****

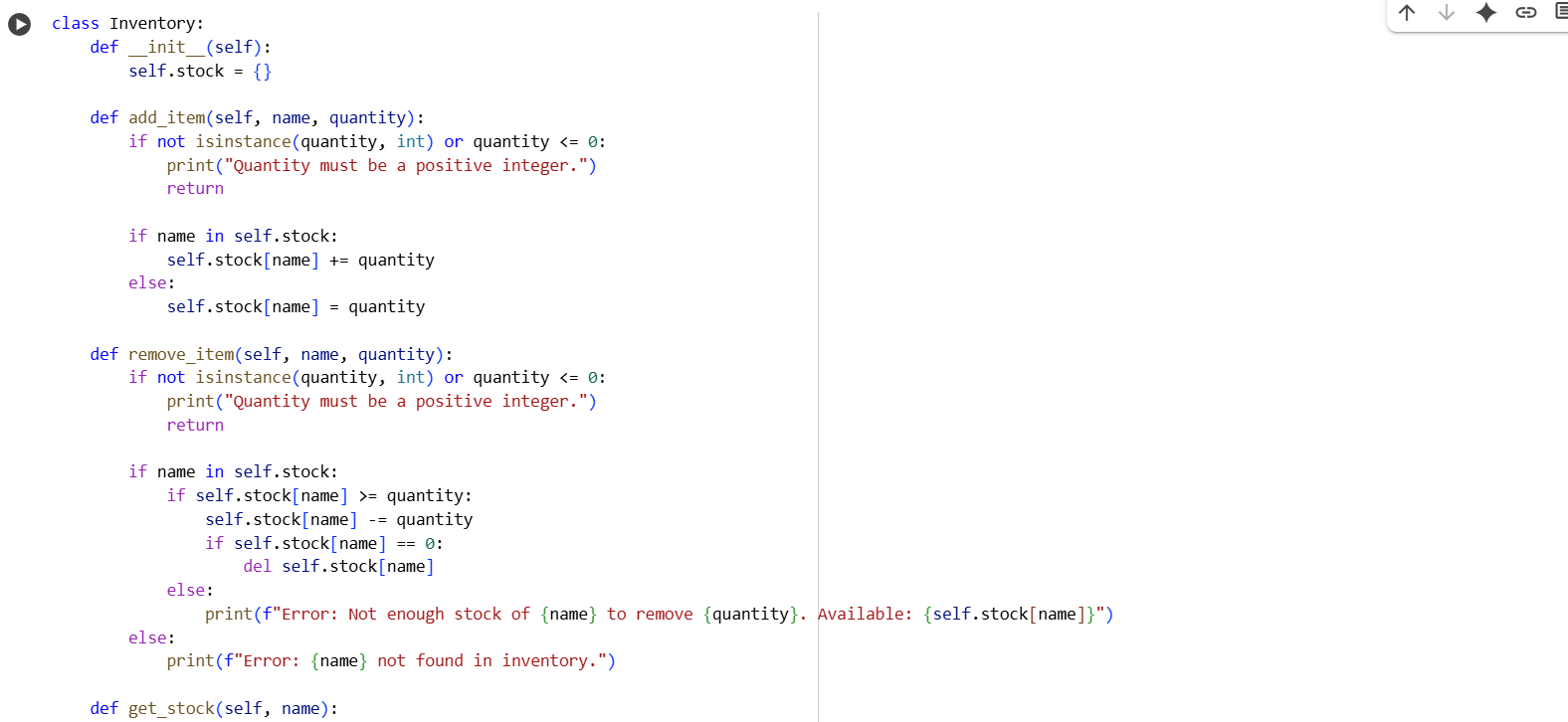
**Observation:**

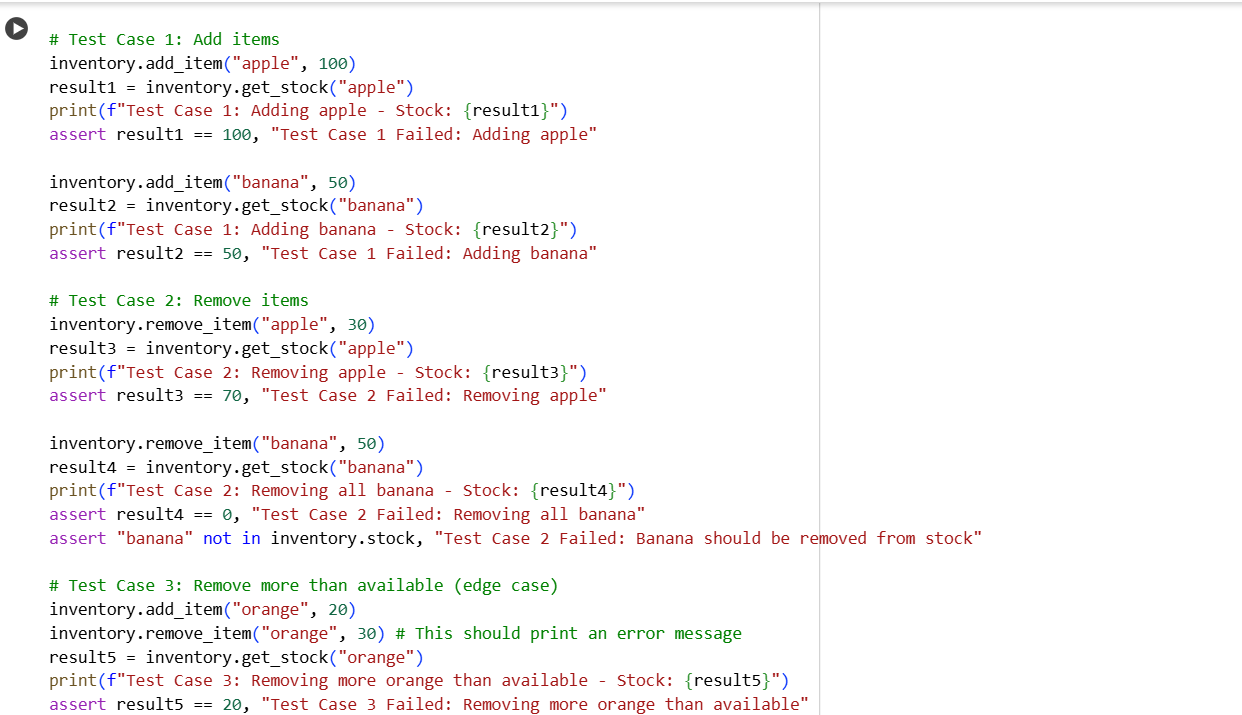
I observed that the is\_anagram(str1, str2) function correctly identified whether two strings are anagrams by ignoring case, spaces, and punctuation. It also handled edge cases such as empty strings and identical words properly. All AI-generated assert test cases passed successfully, confirming that my implementation works as expected.

**#Task 4:**

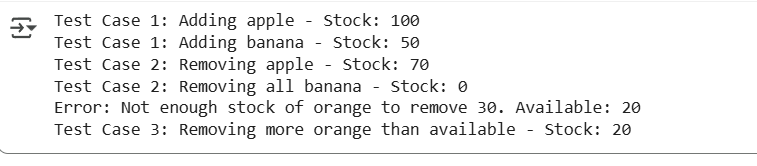
**Prompt:** Create a Python class Inventory with methods add\_item(name, quantity), remove\_item(name, quantity), and get\_stock(name) to manage stock, and generate at least 3 assert test cases to verify adding, removing, and checking stock, including edge cases like removing more than available or checking an item not in inventory.

**Code:**

****

****

**Output:**

****

**Observation:**

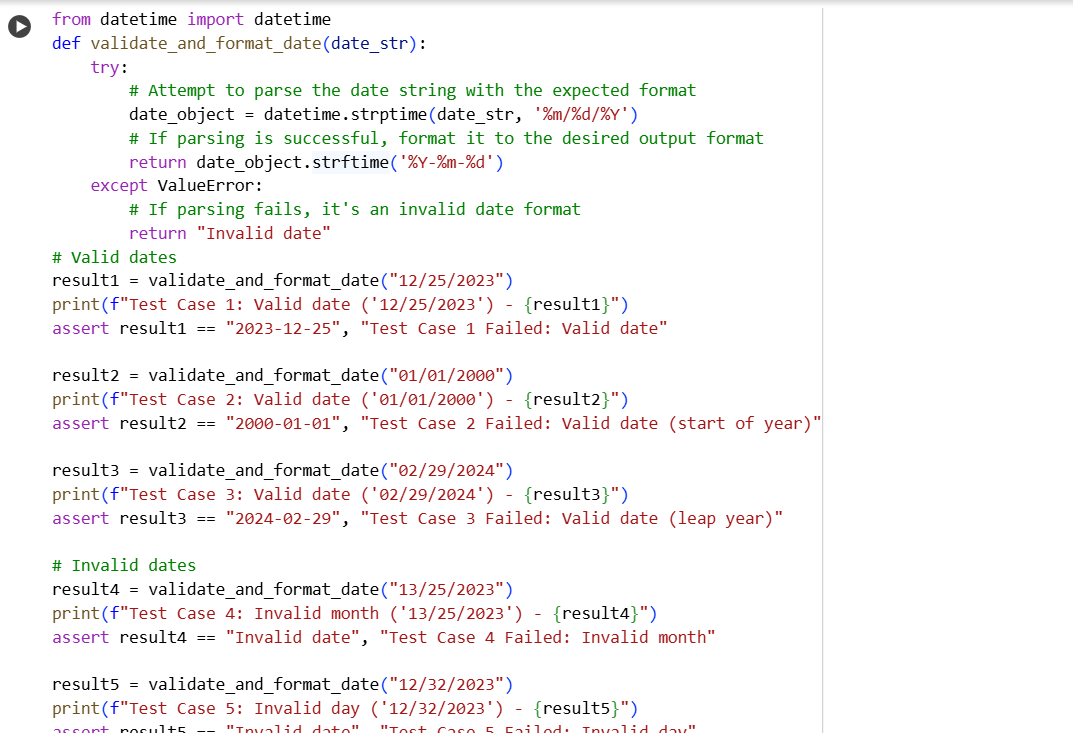
I observed that the Inventory class correctly managed stock by allowing me to add and remove items and accurately check their quantities. It also handled edge cases, such as removing more than the available stock or checking items not yet added, without errors. All AI-generated assert test cases passed successfully, confirming that my implementation works as expected.

**#Task5:**

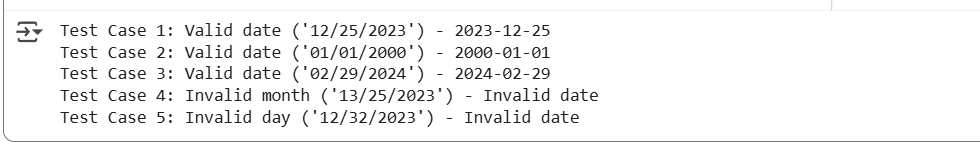
**Prompt:**

Write a Python function validate\_and\_format\_date(date\_str) that checks whether a given date is in the 'MM/DD/YYYY' format, handles invalid dates correctly, and converts valid dates to the 'YYYY-MM-DD' format. Also, generate at least 3 assert test cases to verify the function, including valid dates, invalid dates, and edge cases like leap years or incorrect formats

**Code:**

****

**Output:**

****

**Observation:**

I observed that the validate\_and\_format\_date(date\_str) function correctly validated dates in the "MM/DD/YYYY" format and converted them to "YYYY-MM-DD" for valid inputs. It also handled invalid dates, incorrect formats, and edge cases like leap years properly. All AI-generated assert test cases passed successfully, confirming that my implementation works as expected.