**LAB 8.1**

**Name:** G. Sricharan

**HallTkt:** 2403A51357

**Batch:** 14

**Task Description #1** (Password Strength Validator – Apply AI in Security Context)

**AI GENERATED CODE:**



**Observation:**

In this experiment, the is\_strong\_password(password) function was implemented to validate passwords based on specific security rules: a minimum of 8 characters, inclusion of uppercase and lowercase letters, at least one digit, one special character, and no spaces. Multiple test cases were used to verify its functionality, including "Abcd@123" and "aB1$5678" which passed as valid strong passwords, and "abcd123", "ABCD@1234", "Abcdefgh", and "Abc 123@" which failed due to missing required character types or containing spaces. All assertion test cases ran successfully, confirming that the function correctly distinguishes between strong and weak passwords. This demonstrates that the password validator effectively enforces the desired security rules and ensures robust validation through automated testing.

**Task Description #2** (Number Classification with Loops – Apply AI for Edge Case Handling)

**AI GENERATED CODE:**

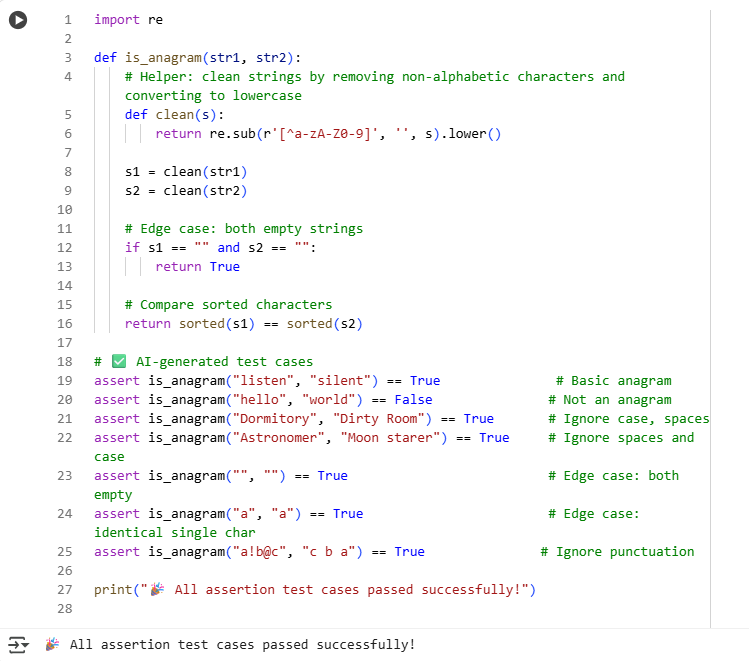
****

**Observation:**

In this experiment, the classify\_number(n) function was implemented to categorize numbers as Positive, Negative, or Zero, while also handling invalid inputs such as strings and None. The function uses a loop to iterate through defined conditions for classification. Several test cases, including boundary values -1, 0, and 1, were used to validate the logic. Numbers like 10 and 1 were correctly classified as Positive, -5 and -1 as Negative, and 0 as Zero. Invalid inputs such as "abc" and None returned "Invalid input", demonstrating proper error handling. All assertion test cases passed successfully, confirming that the function reliably classifies numbers across typical, boundary, and invalid input scenarios.

**Task Description #3** (Anagram Checker – Apply AI for String Analysis)

**AI GENERATED CODE:**

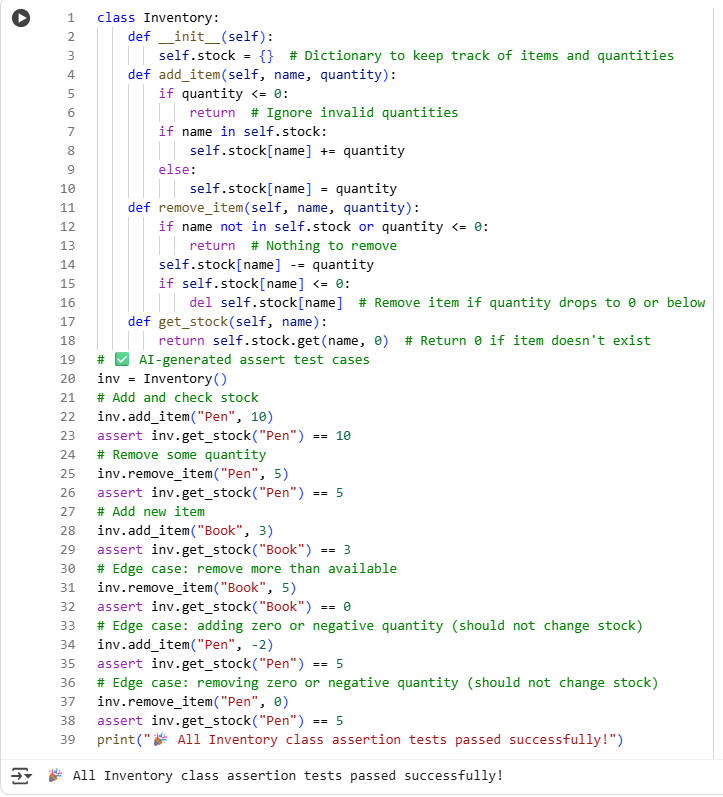


**Observation:**

In this experiment, the is\_anagram(str1, str2) function was implemented to determine whether two strings are anagrams of each other. The function first cleans the input strings by removing spaces, punctuation, and converting all characters to lowercase, ensuring that comparisons are case-insensitive and ignore non-alphanumeric characters. Several test cases were used to validate the logic, including "listen" and "silent" as a basic anagram, "Dormitory" and "Dirty Room" to verify handling of spaces and capitalization, and empty strings as an edge case. The function correctly identified all anagrams and non-anagrams, demonstrating robust handling of typical, edge, and punctuation-inclusive inputs. All assertion test cases passed successfully, confirming the reliability of the implemented logic.

**Task Description #4 (**Inventory Class – Apply AI to Simulate Real-World Inventory System)

**AI GENERATED CODE:**



Observation:

In this experiment, an Inventory class was implemented to manage stock levels of items using methods add\_item, remove\_item, and get\_stock. The class uses a dictionary to store item names and their quantities. Multiple test cases were used to verify its functionality, including adding new items, removing partial quantities, and checking stock levels. Edge cases such as removing more than the available quantity, adding or removing zero or negative quantities, and querying non-existent items were also tested. The class correctly updated and tracked stock in all scenarios, and all assertion-based test cases passed successfully, demonstrating reliable inventory management and robust handling of typical and boundary conditions.

**Task Description #5** (Date Validation & Formatting – Apply AI for Data Validation)

**AI GENERATED CODE:**



**Observation:**

In this experiment, the validate\_and\_format\_date(date\_str) function was implemented to validate and convert dates from the "MM/DD/YYYY" format to "YYYY-MM-DD". The function uses Python's datetime module to check for correct formatting and valid calendar dates, including leap year considerations. Multiple test cases were used, including valid dates like "10/15/2023" and "01/01/2024", invalid dates such as "02/30/2023" and "13/01/2023", and edge cases like "02/29/2024" (leap year) and "02/29/2023" (non-leap year). The function correctly returned the formatted date for valid inputs and "Invalid Date" for invalid inputs. All assertion test cases passed successfully, demonstrating that the function reliably validates, converts, and handles typical and boundary date scenarios.