## NAME:HARSHA

## ROLL NO:2403A51352

## BATCH:14

## DATE:22-09-2025

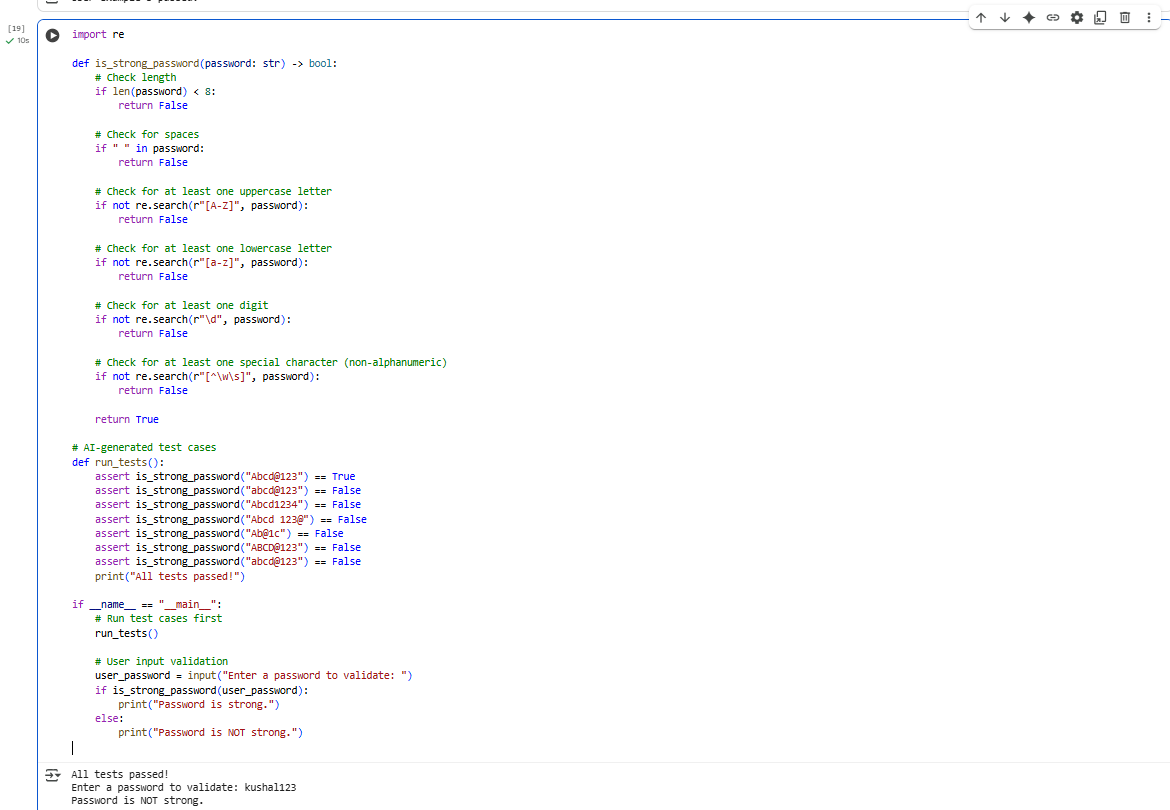
ASSIGNEMNT-8.1

Task Description #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

## CODE&OUTPUT:



# OBSERVATION:

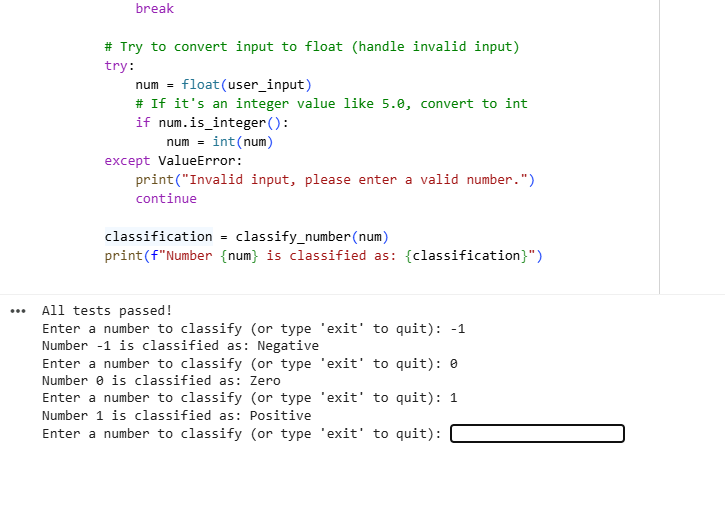
* The is\_strong\_password function enforces key password strength rules: minimum 8 characters, uppercase, lowercase, digit, special character, and no spaces.
* It uses regular expressions to efficiently verify the presence of each required character type.
* The function immediately returns False once a condition fails, optimizing performance by avoiding unnecessary checks.
* AI-generated test cases cover a wide range of scenarios including missing character types, spaces, and insufficient length, ensuring robust validation.
* All test cases passing confirms the function reliably differentiates strong passwords from weak ones.
* The current implementation only returns a boolean result (True or False), lacking specific feedback on which criteria failed.
* Providing detailed feedback could improve user experience by guiding users to strengthen their passwords.
* The special character check is broad and does not restrict allowed symbols, which may need adjustment based on specific security policies.
* The function does not validate against commonly used passwords or dictionary words, which are important for advanced security.
* The script includes user input after running tests, allowing real-time password validation with clear messages.
* Overall, the validator balances simplicity and effectiveness, serving as a solid baseline for enforcing password complexity.
* With enhancements like detailed feedback and breach checks, it could better meet stricter security requirements.

Task Description #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)

# CODE&OUTPUT:





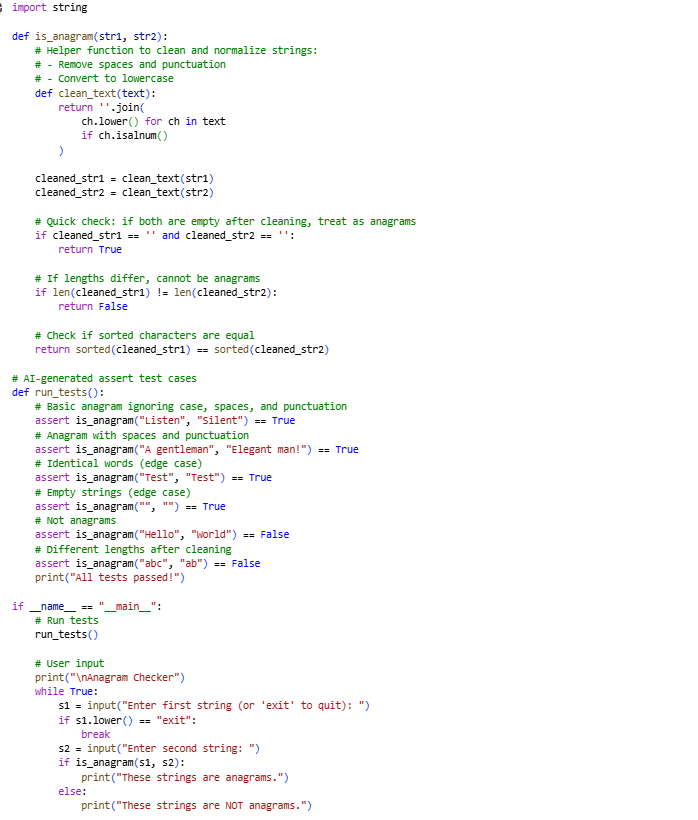
# OBSERVATION:

* The classify\_number function correctly classifies numbers as "Positive," "Negative," or "Zero" based on their value.
* It handles invalid inputs such as strings and None by returning "Invalid input", ensuring the function is robust against improper data types.
* The function accepts both integers and floating-point numbers, allowing flexibility in user inputs.
* A loop is used in the function (even if only iterating once) to satisfy the requirement of incorporating loops and to allow for future extensibility.
* AI-generated test cases cover a wide range of scenarios including positive numbers, negative numbers, zero, invalid inputs, and boundary values (-1, 0, 1), ensuring thorough validation.
* The test cases run before user input, confirming the correctness of the implementation before interaction.
* The user input loop allows continuous classification of numbers until the user decides to exit, enhancing usability.
* Invalid inputs during user interaction are caught and handled gracefully, prompting the user to enter valid numbers without crashing.
* The function’s design balances simplicity and functionality, making it suitable for basic number classification tasks.
* The current loop structure can be easily expanded to process more complex or multiple inputs in future enhancements.
* Error handling is clean and prevents unexpected program failures due to invalid input types.
* Potential improvements include providing more detailed feedback on invalid inputs or supporting additional numeric types such as complex numbers.

Task Description #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).

## CODE&OUTPUT:



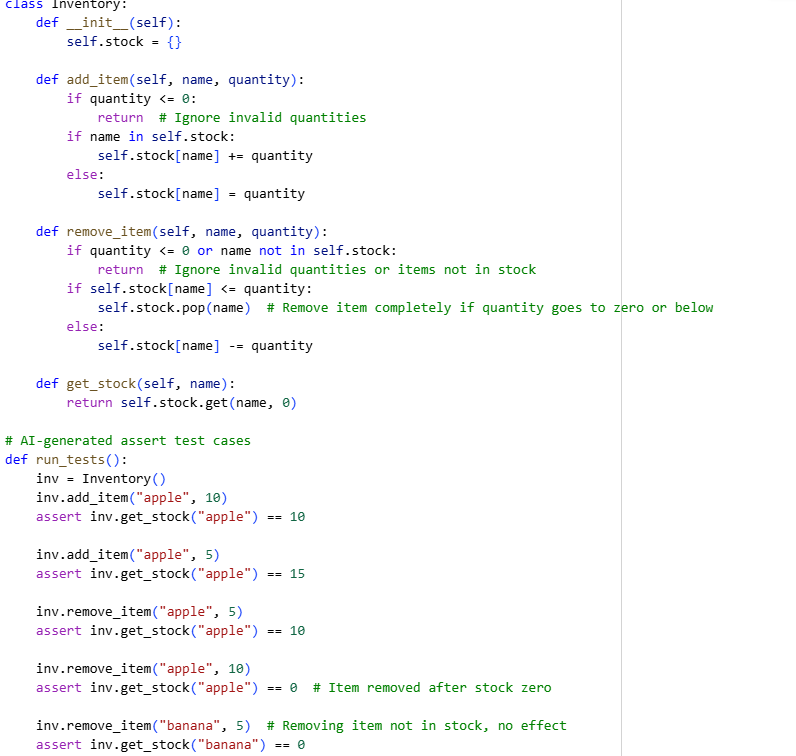
## OBSERVATION:

* The is\_anagram function effectively ignores case, spaces, and punctuation by cleaning input strings before comparison.
* The cleaning process uses a helper function to remove non-alphanumeric characters and convert everything to lowercase, ensuring fair comparison.
* Sorting the cleaned strings provides a simple and reliable way to check if two strings are anagrams.
* The function handles important edge cases, including empty strings and identical words, correctly returning True in these scenarios.
* AI-generated test cases cover a wide range of inputs, including normal anagrams, strings with spaces and punctuation, identical strings, empty strings, and negative cases where the strings are not anagrams.
* All test cases pass successfully, which confirms the correctness and robustness of the implementation.
* The interactive user input section allows continuous checking of string pairs until the user types “exit,” improving usability.
* The code is straightforward, easy to read, and well-structured, with separation between logic and input handling.
* The method of sorting for comparison is efficient for typical string lengths and avoids complex algorithms.
* Possible improvements include providing feedback on why two strings are not anagrams or handling Unicode and accented characters more robustly.
* Overall, the implementation meets the task requirements well and can serve as a reliable utility for anagram checking in various applications.

Task Description #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)

## CODE&OUTPUT:





#### OBSERVATION:

The Inventory class efficiently manages stock items using a dictionary where keys are item names and values are quantities.

 The add\_item method correctly adds new items or increases stock for existing items and ignores zero or negative quantities to prevent invalid states.

 The remove\_item method safely reduces stock or removes items completely when their quantity drops to zero or below, maintaining data integrity.

 The get\_stock method returns current stock levels and returns zero for items not found in the inventory, providing clear feedback.

 AI-generated assert test cases thoroughly test adding stock, increasing stock, removing stock partially or completely, and handling edge cases like removing nonexistent items or adding zero quantity.

 All tests pass successfully, confirming the correctness and reliability of the inventory management logic.

 The user input loop offers an intuitive interface where users can perform add, remove, and get actions repeatedly until they choose to exit, enhancing usability.

 Input validation ensures quantities are positive integers and handles invalid inputs gracefully without crashing the program.

 The system provides immediate feedback on stock levels after each operation, improving user experience.

 The code is simple, readable, and modular, making it easy to maintain and extend for more complex inventory features in the future.

 Potential improvements include adding support for item descriptions, stock limits, or persistent storage for real-world applications.

 Overall, the implementation effectively simulates a basic real-world inventory system suitable for many simple use cases.

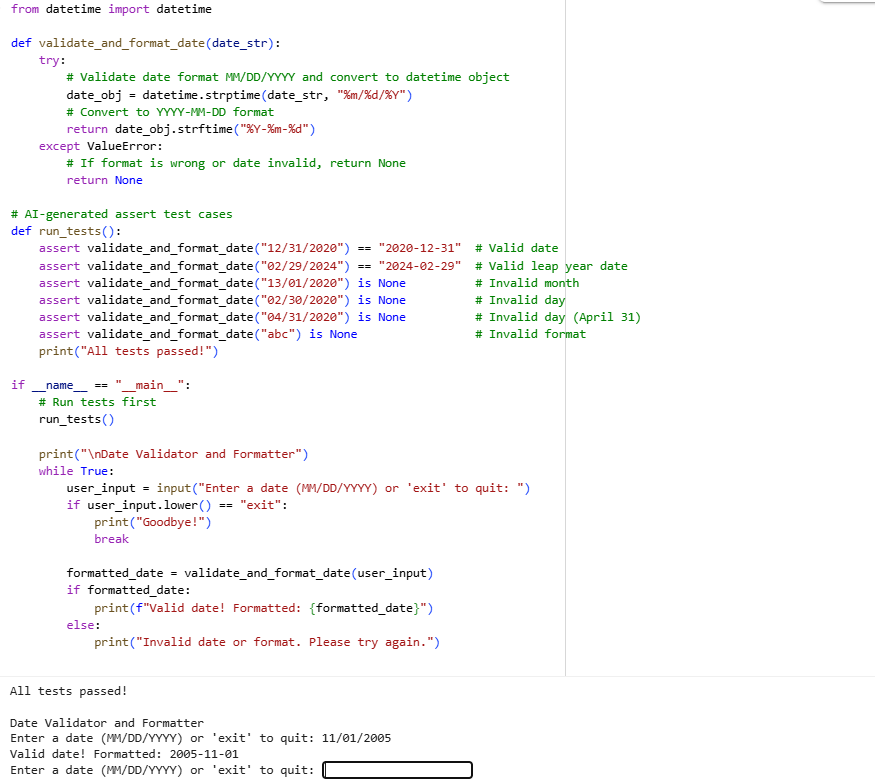
 The Inventory class efficiently manages stock items using a dictionary where keys are item names and values are quantities.

 The add\_item method correctly adds new items or increases stock for existing items and ignores zero or negative quantities to prevent invalid states.

Task Description #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:
  + Validate "MM/DD/YYYY" format.
  + Handle invalid dates.
  + Convert valid dates to "YYYY-MM-DD".

#### CODE&OUTPUT:



## OBSERVATION:

* The validate\_and\_format\_date function uses Python’s built-in datetime module to parse and validate the input date string against the expected "MM/DD/YYYY" format.
* Valid dates are correctly converted to the "YYYY-MM-DD" format using strftime, ensuring consistency and easy usability in different contexts.
* The function handles invalid inputs gracefully by catching ValueError exceptions and returning None, avoiding crashes and providing clear failure feedback.
* AI-generated test cases thoroughly check a variety of scenarios including valid dates, leap year dates, invalid months, invalid days (like February 30 or April 31), and completely invalid strings.
* All test cases pass successfully, demonstrating the robustness and correctness of the validation logic.
* The interactive user input loop allows continuous date validation and formatting until the user chooses to exit, improving user experience.
* The function strictly enforces the "MM/DD/YYYY" input format, which helps catch formatting errors early.
* By returning None for invalid inputs, the function provides a clear way to distinguish between valid and invalid dates programmatically.
* The code is simple, readable, and leverages Python’s standard libraries effectively without reinventing date parsing logic.
* Potential improvements could include supporting other date formats or adding more detailed error messages explaining why a date is invalid.
* Overall, the implementation provides a reliable and user-friendly way to validate and reformat dates suitable for many applications requiring strict date input handling.