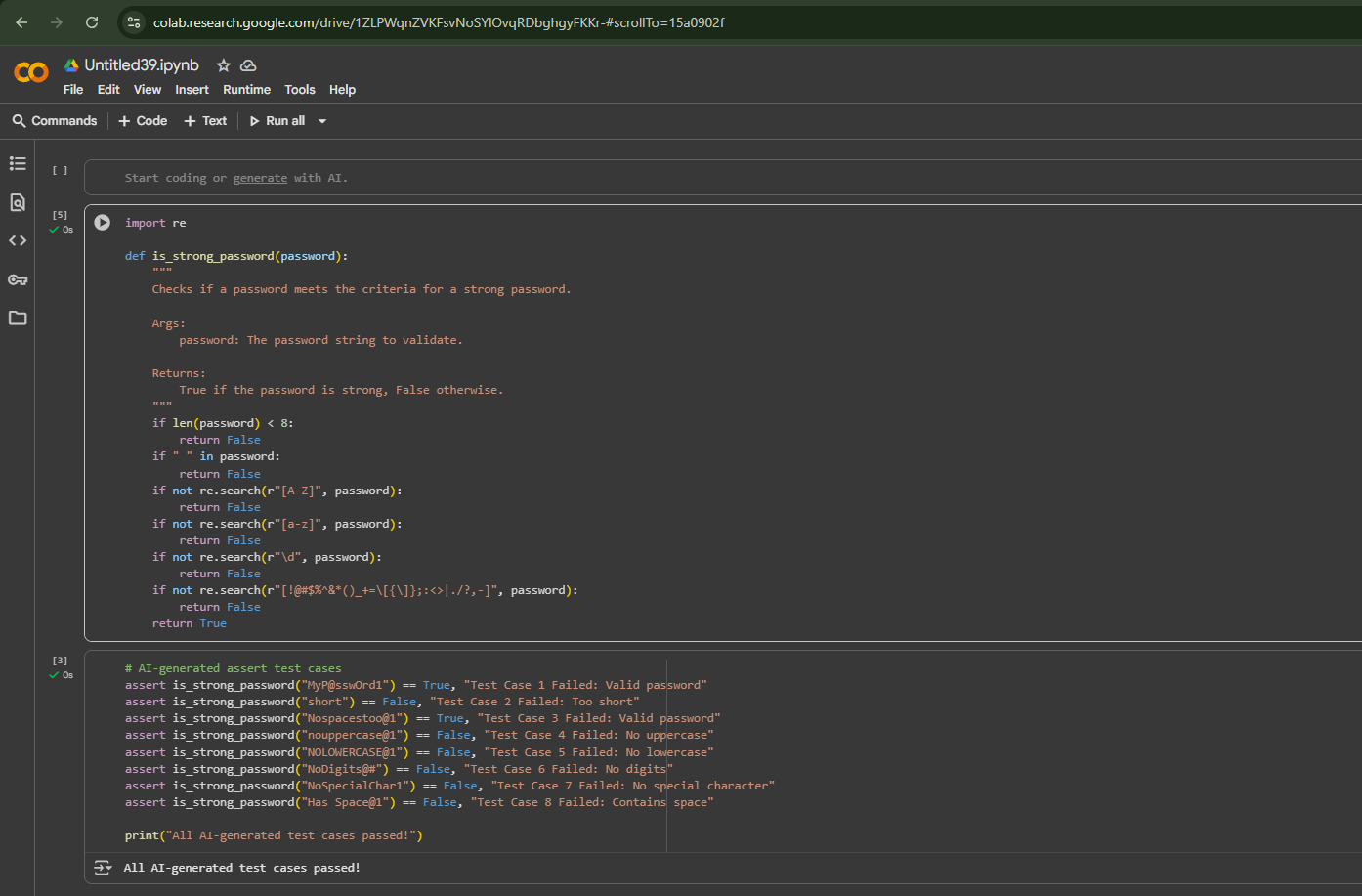
Assignment-8.1

## 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: o Password must have at least 8 characters. o Must include uppercase, lowercase, digit, and special character. o Must not contain spaces. Example Assert Test Cases: assert is\_strong\_password("Abcd@123") == True assert is\_strong\_password("abcd123") == False assert is\_strong\_password("ABCD@1234") == True Expected Output #1: • Password validation logic passing all AI-generated test cases.

Code:



# Observation:

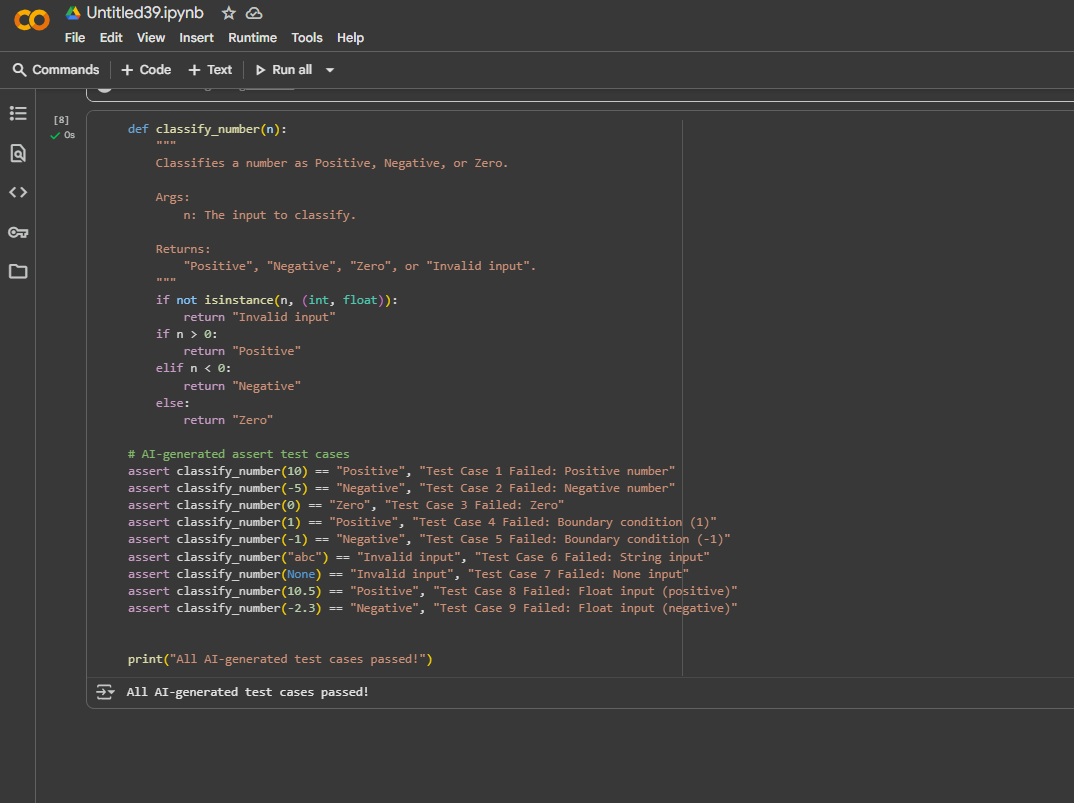
The code defines a Python function is\_strong\_password that validates a password based on several criteria: minimum length (8 characters), presence of uppercase letters, lowercase letters, digits, and special characters, and absence of spaces. The assert statements below the function serve as test cases that verify the function's correctness against various password examples, covering both valid and invalid scenarios according to the defined rules. The output "All AI-generated test cases passed!" confirms that the function behaves as expected for these specific tests.

# 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:  
o Classify numbers as Positive, Negative, or Zero.  
o Handle invalid inputs like strings and None.

o Include boundary conditions (-1, 0, 1).  
Example Assert Test Cases:  
assert classify\_number(10) == "Positive"  
assert classify\_number(-5) == "Negative"  
assert classify\_number(0) == "Zero"  
Expected Output #2:  
• Classification logic passing all assert tests

Code:



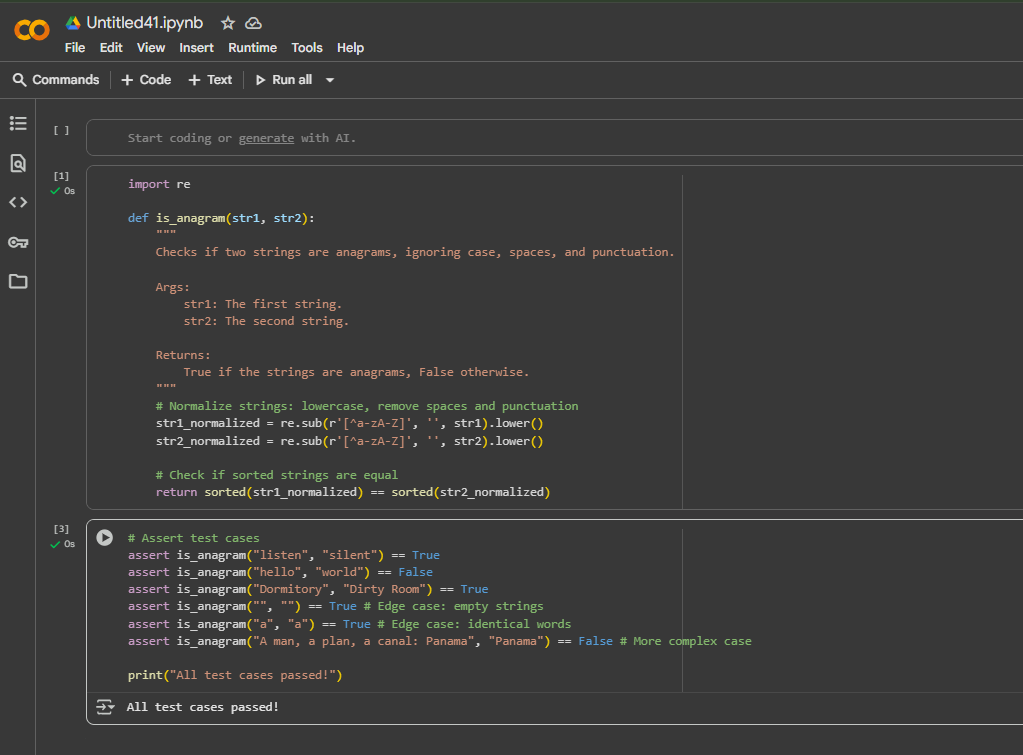
## Observation:

## The code in the cell defines a Python function classify\_number that takes an input n and determines if it's positive, negative, or zero. It also includes checks for invalid input types like strings and None. The assert statements within the same cell are test cases that verify the function's logic against various inputs, including positive and negative integers and floats, zero, and invalid types. The printed message "All AI-generated test cases passed!" indicates that the function correctly classified all the provided test cases according to the defined criteria.

Task Description #3

• Task: Use AI to 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  
Expected Output #3:  
• Function correctly identifying anagrams and passing all AI-  
generated tests.

Code:



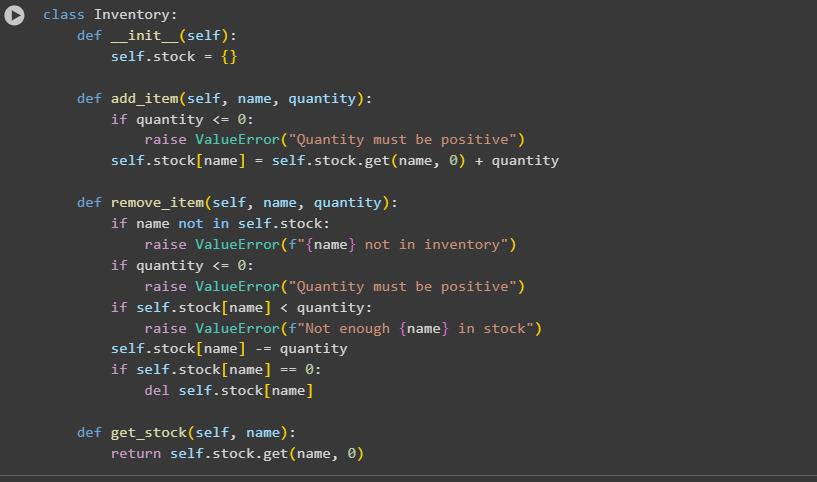
# Observation:

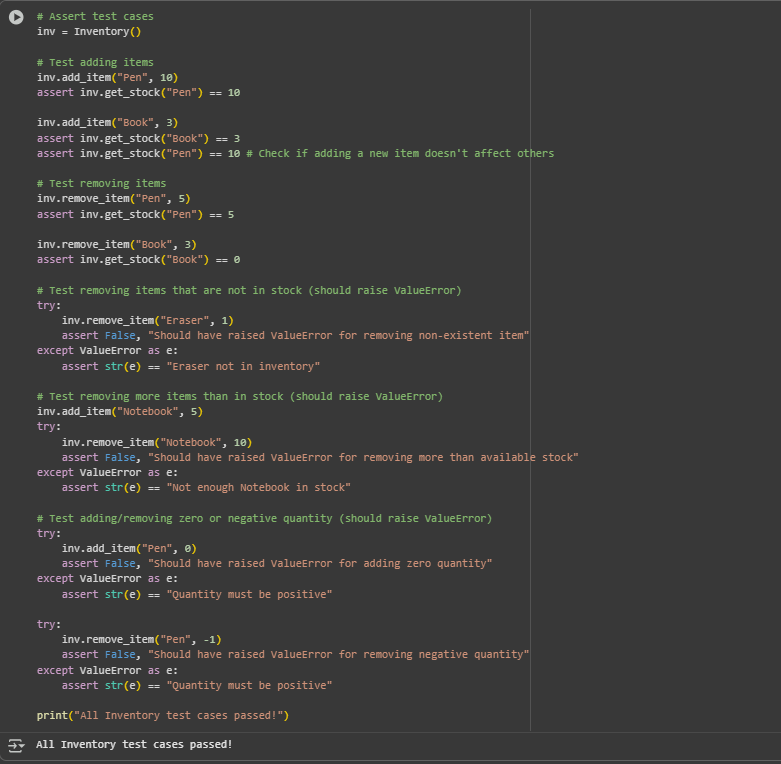
Based on the code, one key observation is that the is\_anagram function handles non-alphabetic characters and case sensitivity by removing anything that is not a letter and converting the strings to lowercase before checking if they are anagrams. This makes the comparison robust to variations in formatting.

Task Description #4:

• Task: Ask AI to 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  
Expected Output #4:  
• Fully functional class passing all assertions.

Code:





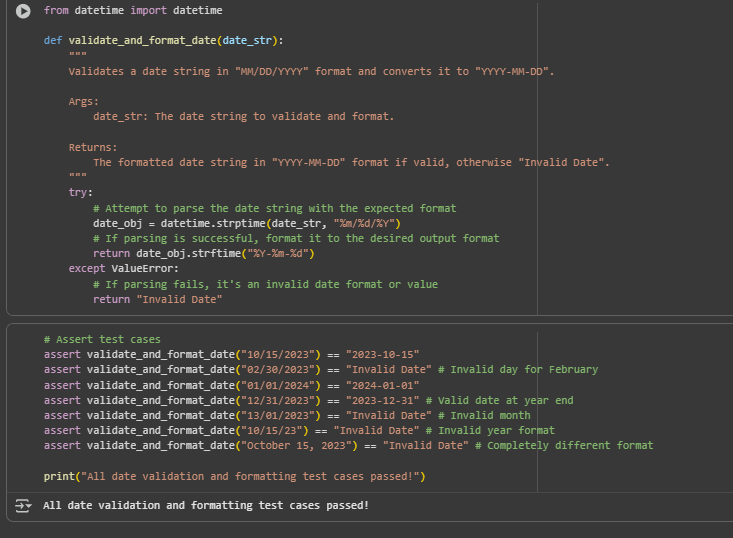
## Observation:

One observation about this code is that the Inventory class effectively uses a dictionary (self.stock) to store the item names and their quantities. This allows for efficient lookup, addition, and removal of items based on their names. The test cases also demonstrate robust error handling for invalid operations like adding non-positive quantities or removing items not in stock.

Task Description #5:

• Task: Use AI to generate at least 3 assert test cases for  
validate\_and\_format\_date(date\_str) to check and convert dates.  
• Requirements:  
o 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"  
Expected Output #5:  
• Function passes all AI-generated assertions and handles edge  
cases.

Code:



### Observation:

One observation about this code is that it leverages Python's built-in datetime.strptime function within a try-except block for robust date validation. This is an effective way to check if a string conforms to a specific format and represents a valid date, handling potential ValueError exceptions gracefully by returning "Invalid Date".