ASSIGNMENT 8.1

V. RAVI TEJA

2303A51942

BATCH 12

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

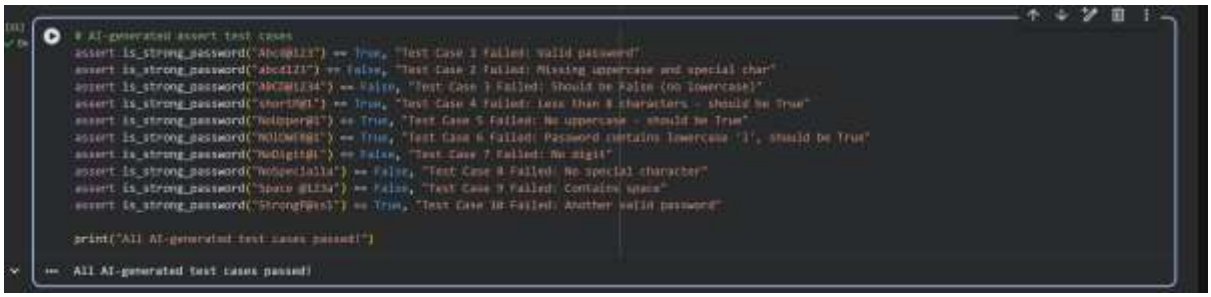
assert is_strong_password("ABCD@1234") == True

Expected Output #1:

• Password validation logic passing all AI-generated test cases.

CODE:

OUTPUT:



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"

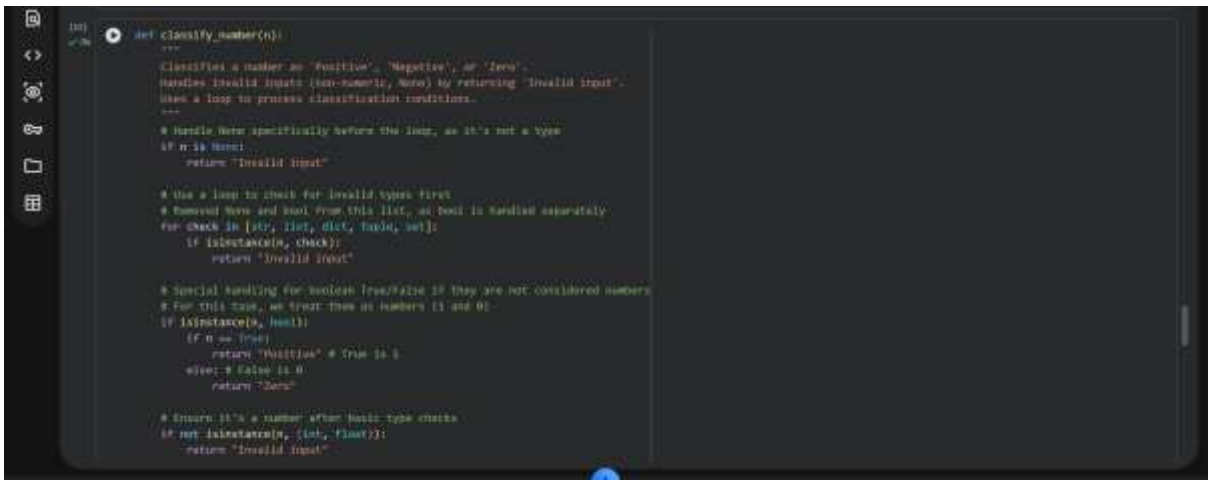
assert classify_number(-5) == "Negative"

assert classify_number(0) == "Zero"

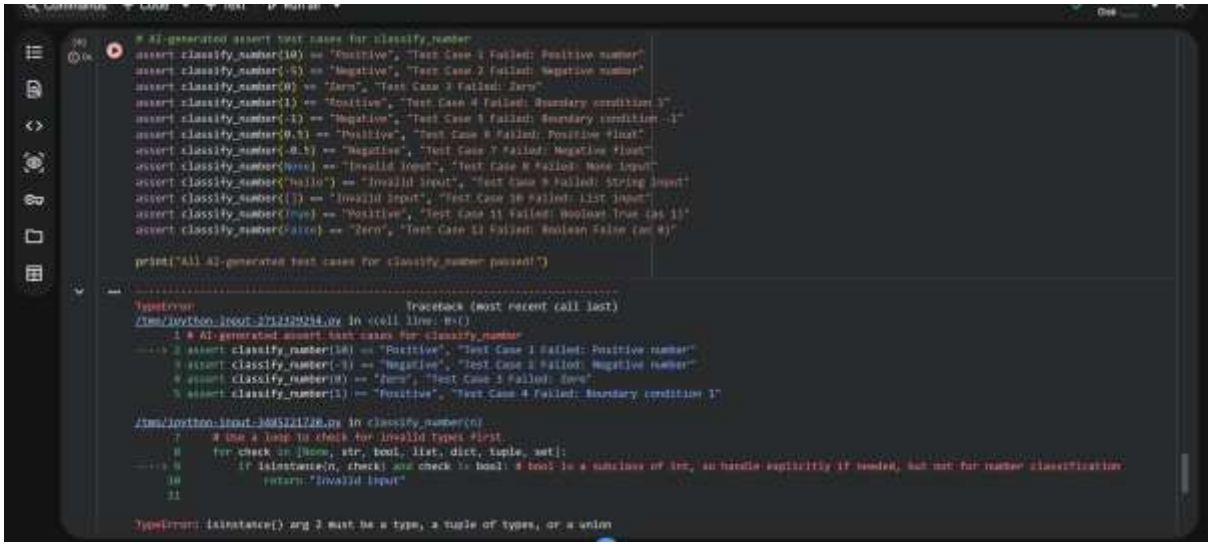
Expected Output #2:

• Classification logic passing all assert tests.

CODE:



OUTPUT:



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:

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:

```
import re

def is_anagram(str1, str2):
    """
    Checks if two strings are anagrams of each other.
    Ignores case, spaces, and punctuation.
    Handles empty strings and identical words.
    """

    def normalize_string(s):
        # Convert to lowercase
        s = s.lower()
        # Remove spaces and punctuation
        s = re.sub(r'[^a-z0-9]', '', s) # Keep only alphanumeric characters
        return s

    normalized_str1 = normalize_string(str1)
    normalized_str2 = normalize_string(str2)

    # Anagrams must have the same length after normalization
    if len(normalized_str1) != len(normalized_str2):
        return False

    # Sort the normalized strings and compare
    return sorted(normalized_str1) == sorted(normalized_str2)
```

OUTPUT:

```
# AI-generated assert test cases for is_anagram
assert is_anagram("listen", "silent") == True, "Test Case 1 Failed: Basic anagram"
assert is_anagram("hello", "world") == False, "Test Case 2 Failed: Not an anagram"
assert is_anagram("Dormitory", "Dirty Room") == True, "Test Case 3 Failed: Anagram with spaces and case"
assert is_anagram("A decimal point", "I'm a dot in place") == True, "Test Case 4 Failed: Anagram with punctuation and spaces"
assert is_anagram("", "") == True, "Test Case 5 Failed: Both empty strings"
assert is_anagram("a", "") == False, "Test Case 6 Failed: One empty string"
assert is_anagram("identical", "identical") == True, "Test Case 7 Failed: Identical words"
assert is_anagram("Debit card", "Bad credit") == True, "Test Case 8 Failed: Anagram with different words, same meaning"
assert is_anagram("openai", "openAI") == True, "Test Case 9 Failed: Same words, different case"
assert is_anagram("Race car", "Car race") == True, "Test Case 10 Failed: Anagram with spaces and different order"

print("All AI-generated test cases for is_anagram passed!")
```

```
All AI-generated test cases for is_anagram passed!
```

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:

o add_item(name, quantity)

o remove_item(name, quantity)

o get_stock(name)

Example Assert Test Cases:

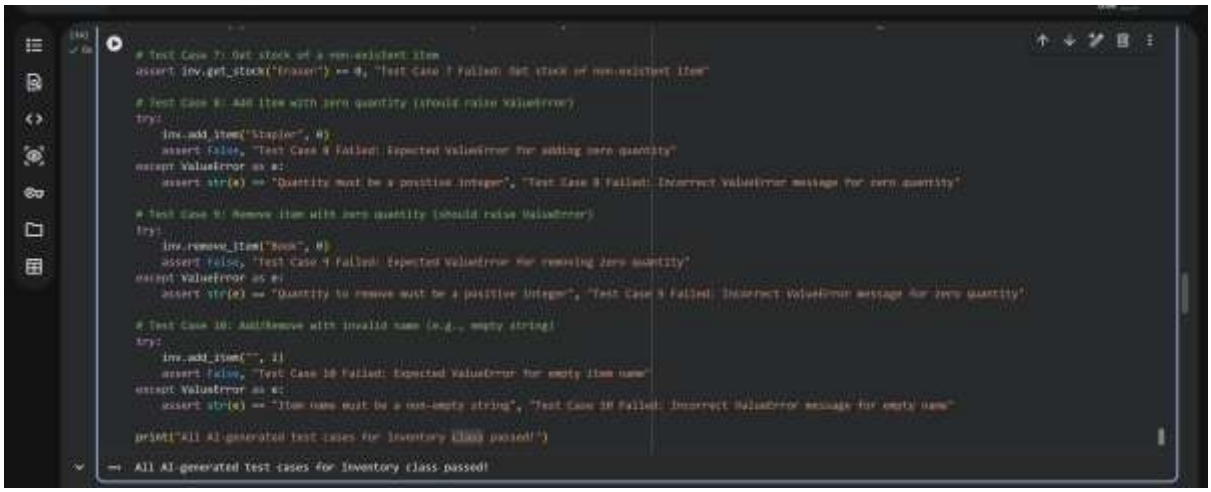
```
inv = Inventory()
inv.add_item("Pen", 10)
assert inv.get_stock("Pen") == 10
inv.remove_item("Pen", 5)
assert inv.get_stock("Pen") == 5
inv.add_item("Book", 3)
assert inv.get_stock("Book") == 3
```

Expected Output #4:

• Fully functional class passing all assertions.

CODE:

OUTPUT:



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:

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:

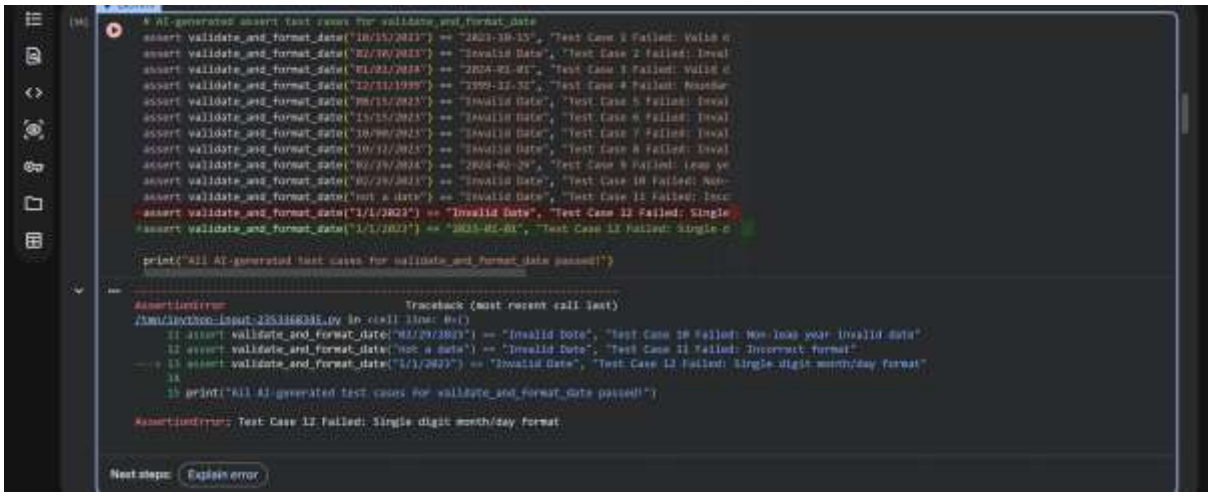
• Function passes all AI-generated assertions and handles edge cases.

CODE:

```
import datetime

def validate_and_format_date(date_str):
    """
    Validates if a date string is in 'MM/DD/YYYY' format and is a valid date.
    If valid, converts it to 'YYYY-MM-DD' format.
    If invalid, returns 'Invalid Date'.
    """
    try:
        # Attempt to parse the date string using the specified format
        dt_object = datetime.datetime.strptime(date_str, "%m/%d/%Y")
        # If parsing is successful, format it to 'YYYY-MM-DD'
        return dt_object.strftime("%Y-%m-%d")
    except ValueError:
        # If parsing fails (due to incorrect format or invalid date),
        # catch the ValueError and return 'Invalid Date'
        return "Invalid Date"
```

OUTPUT:



```
# AI-generated assert test cases for validate_and_format_date
assert validate_and_format_date("10/15/2023") == "2023-10-15", "Test Case 1 Failed: Valid d
assert validate_and_format_date("02/30/2023") == "Invalid Date", "Test Case 2 Failed: Inval
assert validate_and_format_date("01/01/2024") == "2024-01-01", "Test Case 3 Failed: Valid c
assert validate_and_format_date("12/31/1999") == "1999-12-31", "Test Case 4 Failed: Boundar
assert validate_and_format_date("00/15/2023") == "Invalid Date", "Test Case 5 Failed: Inval
assert validate_and_format_date("13/15/2023") == "Invalid Date", "Test Case 6 Failed: Inval
assert validate_and_format_date("10/00/2023") == "Invalid Date", "Test Case 7 Failed: Inval
assert validate_and_format_date("10/32/2023") == "Invalid Date", "Test Case 8 Failed: Inval
assert validate_and_format_date("02/29/2024") == "2024-02-29", "Test Case 9 Failed: Leap ye
assert validate_and_format_date("02/29/2023") == "Invalid Date", "Test Case 10 Failed: Non-
assert validate_and_format_date("not a date") == "Invalid Date", "Test Case 11 Failed: Inco
assert validate_and_format_date("1/1/2023") == "Invalid Date", "Test Case 12 Failed: Single-d
#assert validate_and_format_date("1/1/2023") == "2023-01-01", "Test Case 12 Failed: Single-d

print("All AI-generated test cases for validate_and_format_date passed!")

---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
/tmp/ipython-input-2353368345.py in <cell line: 0>()
     11 assert validate_and_format_date("02/29/2023") == "Invalid Date", "Test Case 10 Failed: Non-leap year invalid date"
     12 assert validate_and_format_date("not a date") == "Invalid Date", "Test Case 11 Failed: Incorrect format"
---> 13 assert validate_and_format_date("1/1/2023") == "Invalid Date", "Test Case 12 Failed: Single-digit month/day format"
     14
     15 print("All AI-generated test cases for validate_and_format_date passed!")

AssertionError: Test Case 12 Failed: Single-digit month/day format

Next steps:  Explain error
```