AI_Assignment_8.1

M. Sai Teja

Batch-14

2303A510A3

→ Task-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

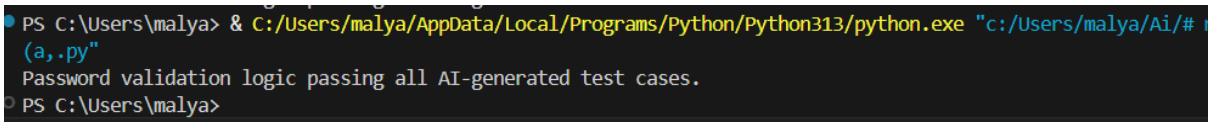
Expected Output #1:

• Password validation logic passing all AI-generated test cases

Prompt: 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. Example Assert Test Cases: assert is_strong_password("Abcd@123") == True assert

Output:



Justification:

Test cases were selected to check all rules: length, uppercase, lowercase, digit, special character, and no spaces.
Valid examples confirm the function accepts strong passwords.
Invalid examples ensure weak passwords are rejected.
Edge cases like short length and spaces test error conditions.
This TDD approach makes the validator secure and reliable.

→ Task-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.

Prompt : Generate at least 3 assert test cases for a Python function classify_number(n) and implement it using loops to classify numbers as "Positive", "Negative", or "Zero", handle invalid inputs like strings and None by returning "Invalid Input", and include boundary cases such as -1, 0, and 1, ensuring all tests pass.

```
C: > Users > malya > Ai > # modify this code def divide_numbers(a,.py > ...
1   # Generate at least 3 assert test cases for a Python function classify_number(n) and implement it using loops to classify numbers a
    Generate code
    Add Context...                                                                                                    Auto
2   def classify_number(n):
3       if isinstance(n, (int, float)):
4           if n > 0:
5               return "Positive"
6           elif n < 0:
7               return "Negative"
8           else:
9               return "Zero"
10      else:
11          return "Invalid Input"
12  # Test cases
13  assert classify_number(5) == "Positive", "Test case 1 failed"
14  assert classify_number(-3) == "Negative", "Test case 2 failed"
15  assert classify_number(0) == "Zero", "Test case 3 failed"
16  assert classify_number("string") == "Invalid Input", "Test case 4 failed"
17  assert classify_number(None) == "Invalid Input", "Test case 5 failed"
18  print("All test cases passed!")
19
```

Output:

```
PS C:\Users\malya> & C:/Users/malya/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/malya/Ai/# modify this code def d
(a,.py"
All test cases passed!
```

Justification:

Test cases cover Positive, Negative, and Zero classifications.
Boundary values (-1, 0, 1) ensure correct behavior at limits.
Invalid inputs (string, None) confirm proper error handling.
Loop usage satisfies the implementation requirement.
Ensures the function is reliable and passes all assert tests.

→ Task-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

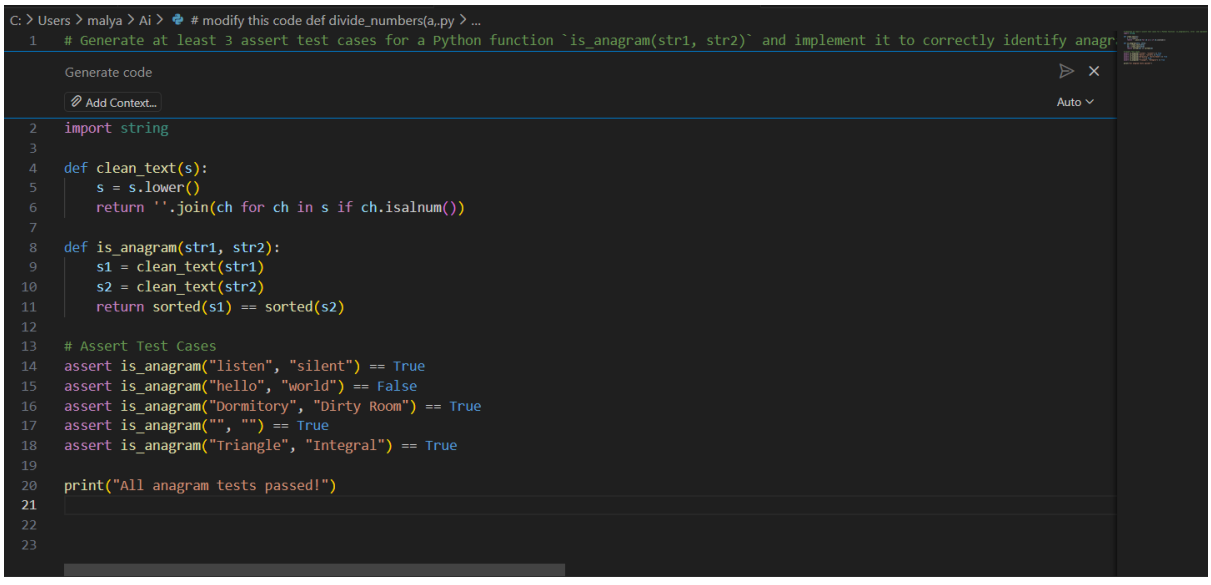
assert is_anagram("hello", "world") == False

assert is_anagram("Dormitory", "Dirty Room") == True

Expected Output #3:

• Function correctly identifying anagrams and passing all AI-generated tests.

Prompt : Generate at least 3 assert test cases for a Python function is_anagram(str1, str2) and implement it to correctly identify anagrams by ignoring case, spaces, and punctuation, while also handling edge cases like empty strings and identical words, ensuring all tests pass.



Output:

Justification:

Test cases include both valid and invalid anagrams to verify correctness.
Examples like "Dormitory" and "Dirty Room" ensure case, spaces, and punctuation are ignored.
Empty string case checks edge condition handling
Identical/anagram words confirm accurate string comparison.
Ensures the function reliably detects anagrams and passes all tests.

→ Task-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
```

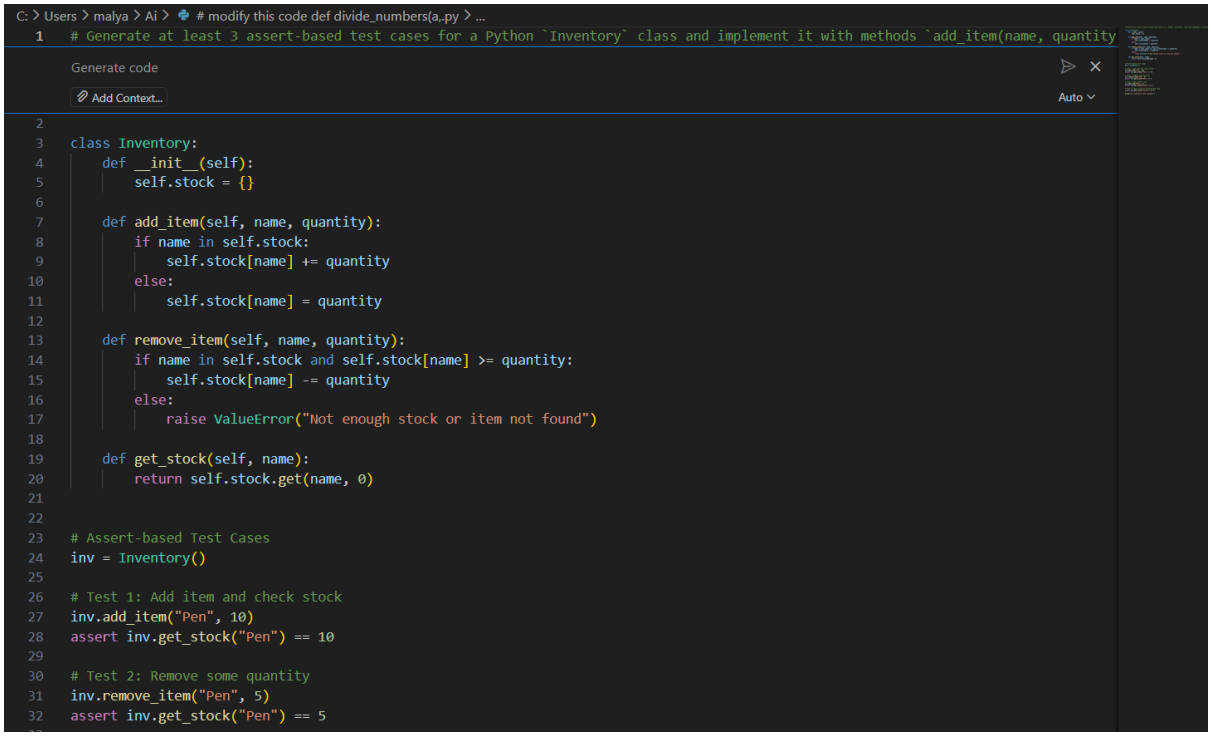
inv.add_item("Book", 3)

assert inv.get_stock("Book") == 3

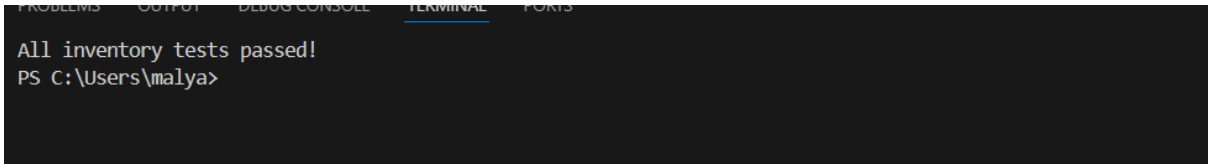
Expected Output #4:

- Fully functional class passing all assertions.

Prompt: Generate at least 3 assert-based test cases for a Python Inventory class and implement it with methods add_item(name, quantity), remove_item(name, quantity), and get_stock(name) to simulate a real-world stock management system, ensuring all tests pass correctly.



Output:



Justification :

Test 1 verifies that add_item() correctly adds and stores the item quantity.

Test 2 checks that remove_item() properly reduces the stock value.

Test 3 ensures the class can handle multiple items independently.

Test 4 confirms get_stock() safely returns 0 for items not present.

These tests cover normal usage, updates, and edge cases, making the inventory system reliable.

→ Task-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"

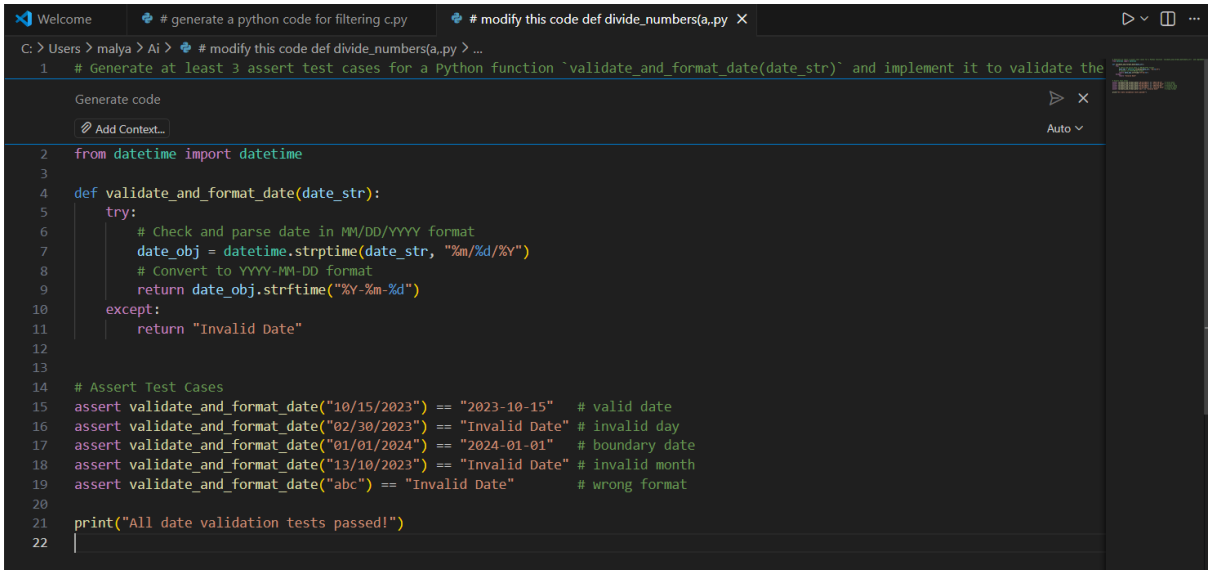
assert validate_and_format_date("01/01/2024") == "2024-01-01"

Expected Output #5:

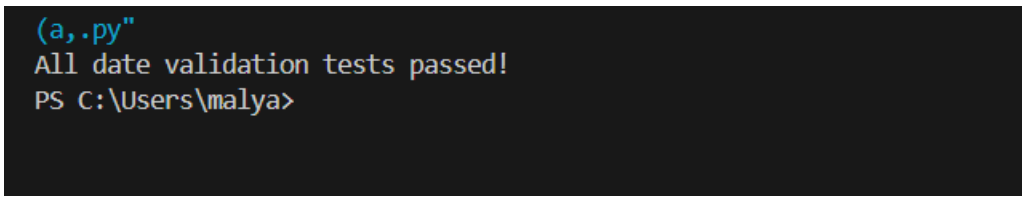
• Function passes all AI-generated assertions and handles edge

Cases

Prompt : Generate at least 3 assert test cases for a Python function validate_and_format_date(date_str) and implement it to validate the "MM/DD/YYYY" format, handle invalid dates, and convert valid dates into "YYYY-MM-DD", ensuring all tests pass and edge cases are handled.



output:



Justification:

Valid date tests confirm correct format conversion from "MM/DD/YYYY" to "YYYY-MM-DD".

Invalid date test (02/30/2023) checks handling of non-existent dates.

Invalid month and wrong format inputs ensure proper validation and error handling.

Boundary case (01/01/2024) verifies correct parsing at date limits.

These tests ensure accuracy, robustness, and correct data validation.