AI ASSISTED CODING

LAB-8.1

PUVATI VARDHINI

2303A51301

Batch-11

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:

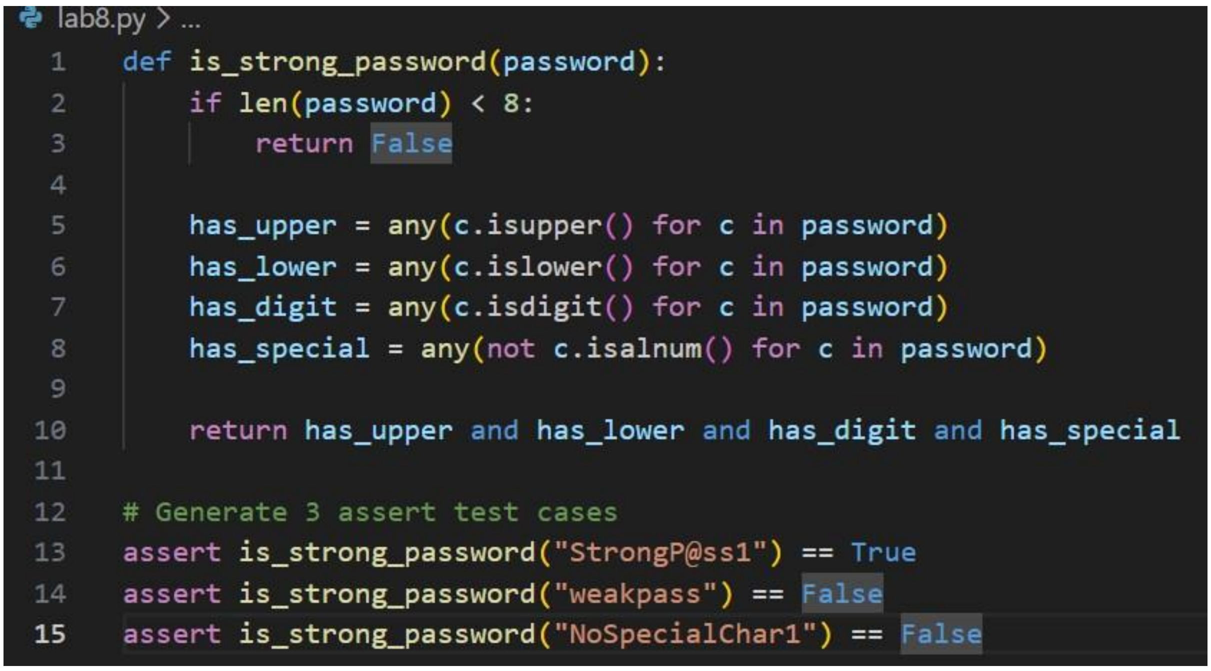
• Password validation logic passing all AI-generated test cases.

Prompt:

def is_strong_password():

Give 3 assert test cases

Given Code:



lab8.py > ...

```python
def is_strong_password(password):
    if len(password) < 8:
        return False

    has_upper = any(c.isupper() for c in password)
    has_lower = any(c.islower() for c in password)
    has_digit = any(c.isdigit() for c in password)
    has_special = any(not c.isalnum() for c in password)

    return has_upper and has_lower and has_digit and has_special

# Generate 3 assert test cases
assert is_strong_password("StrongP@ss1") == True
assert is_strong_password("weakpass") == False
assert is_strong_password("NoSpecialChar1") == False
```

<u>Task Description #2 (Number Classification with Loops – Apply AI for Edge Case Handling)</u>

• 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

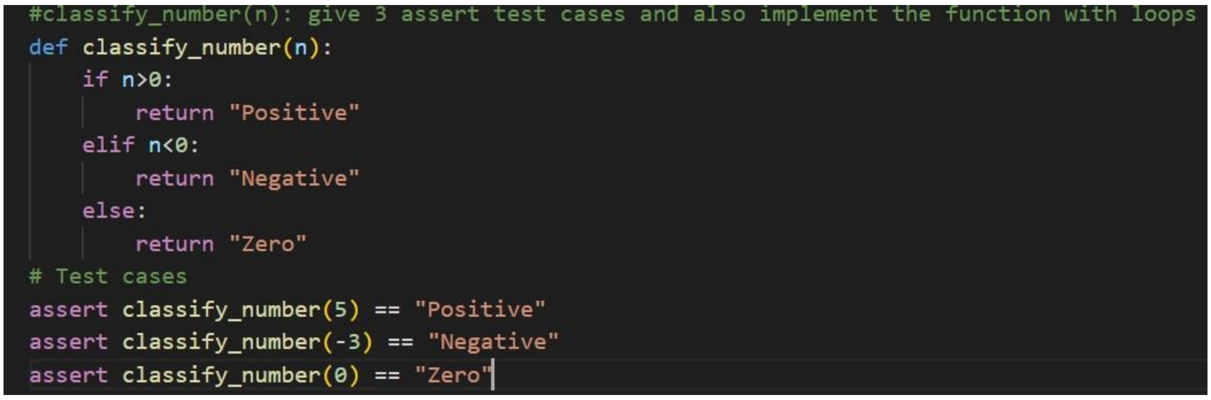
classify_number(-5) == "Negative" assert

classify_number(0) == "Zero"

<u>Prompt:</u>

#classify_number(n): give 3 assert test cases and also implement the function with loops

<u>Given Code:</u>



```python
#classify_number(n): give 3 assert test cases and also implement the function with loops
def classify_number(n):
    if n>0:
        return "Positive"
    elif n<0:
        return "Negative"
    else:
        return "Zero"
# Test cases
assert classify_number(5) == "Positive"
assert classify_number(-3) == "Negative"
assert classify_number(0) == "Zero"
```

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

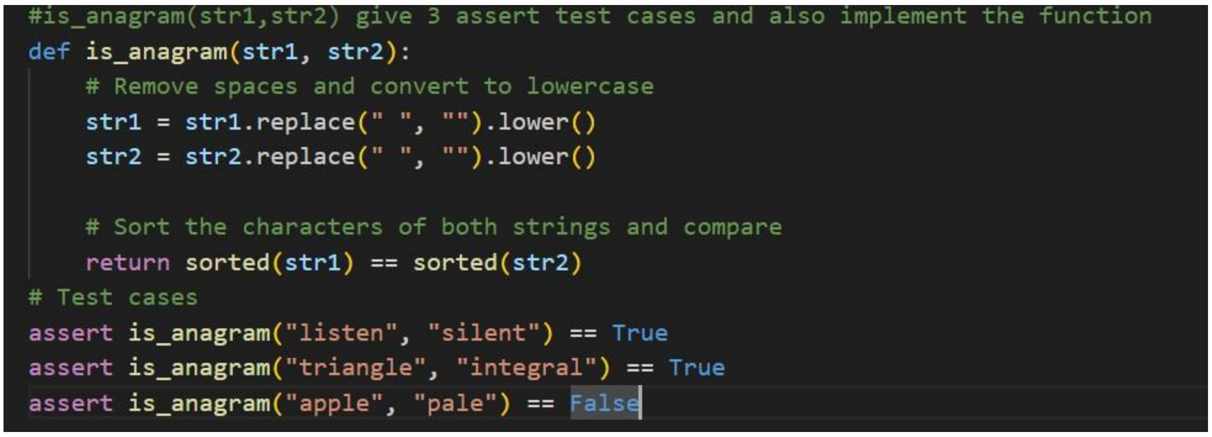
Expected Output #3:

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

Prompt:

#is_anagram(str1,str2) give 3 assert test cases and also implement the function **Given**

Code:



```python
#is_anagram(str1,str2) give 3 assert test cases and also implement the function
def is_anagram(str1, str2):
    # Remove spaces and convert to lowercase
    str1 = str1.replace(" ", "").lower()
    str2 = str2.replace(" ", "").lower()

    # Sort the characters of both strings and compare
    return sorted(str1) == sorted(str2)
# Test cases
assert is_anagram("listen", "silent") == True
assert is_anagram("triangle", "integral") == True
assert is_anagram("apple", "pale") == False
```

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.

Prompt:

"""Give python code for inventory class with stock management methods:

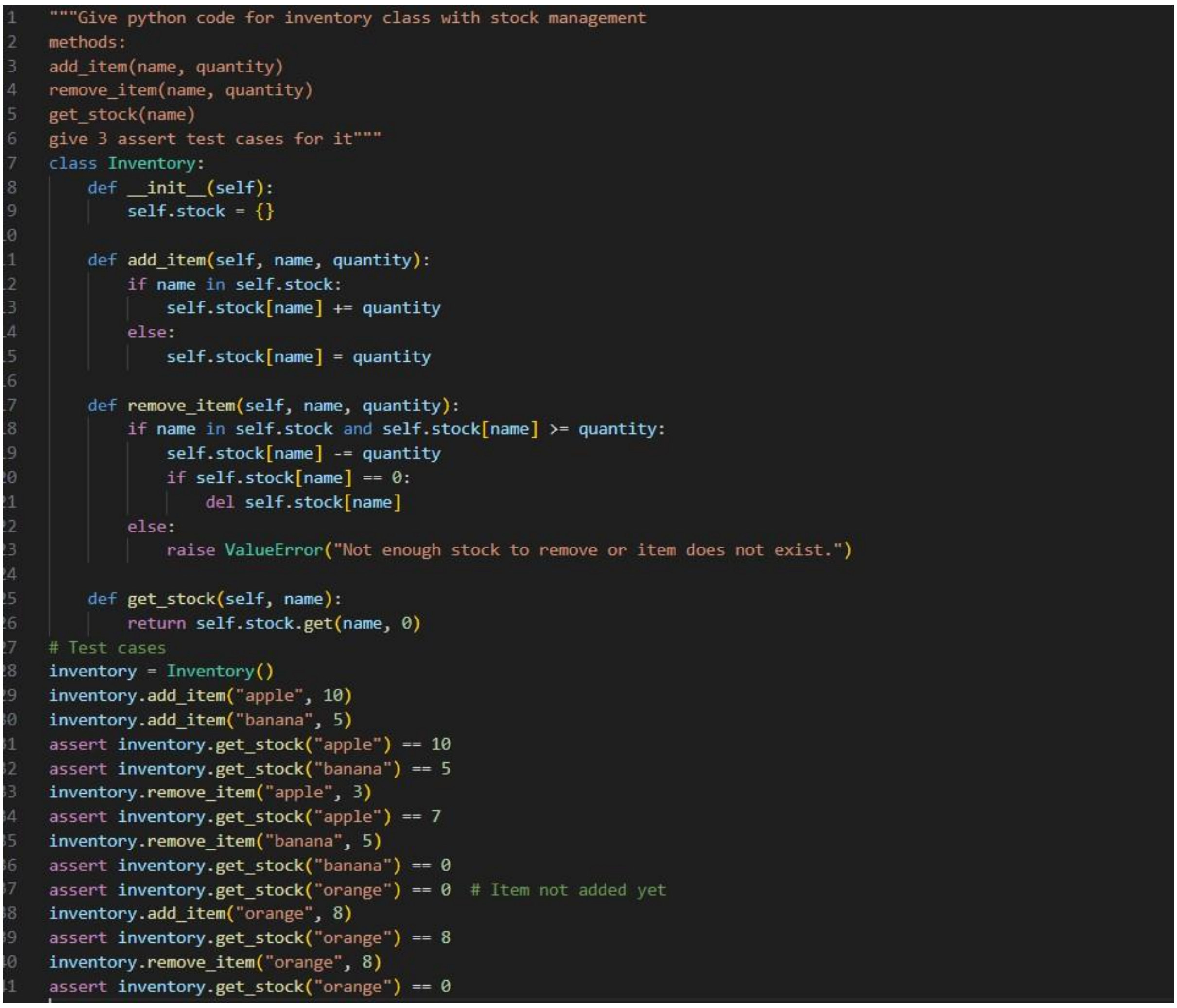
add_item(name,   quantity)

remove_item(name, quantity)

get_stock(name)

give 3 assert test cases for it"""

Given Code:



```python
"""Give python code for inventory class with stock management
methods:
add_item(name, quantity)
remove_item(name, quantity)
get_stock(name)
give 3 assert test cases for it"""
class Inventory:
    def __init__(self):
        self.stock = {}

    def add_item(self, name, quantity):
        if name in self.stock:
            self.stock[name] += quantity
        else:
            self.stock[name] = quantity

    def remove_item(self, name, quantity):
        if name in self.stock and self.stock[name] >= quantity:
            self.stock[name] -= quantity
            if self.stock[name] == 0:
                del self.stock[name]
        else:
            raise ValueError("Not enough stock to remove or item does not exist.")

    def get_stock(self, name):
        return self.stock.get(name, 0)
# Test cases
inventory = Inventory()
inventory.add_item("apple", 10)
inventory.add_item("banana", 5)
assert inventory.get_stock("apple") == 10
assert inventory.get_stock("banana") == 5
inventory.remove_item("apple", 3)
assert inventory.get_stock("apple") == 7
inventory.remove_item("banana", 5)
assert inventory.get_stock("banana") == 0
assert inventory.get_stock("orange") == 0  # Item not added yet
inventory.add_item("orange", 8)
assert inventory.get_stock("orange") == 8
inventory.remove_item("orange", 8)
assert inventory.get_stock("orange") == 0
```

<u>Task Description #5 (Date Validation & Formatting – Apply AI for Data Validation)</u>

• 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

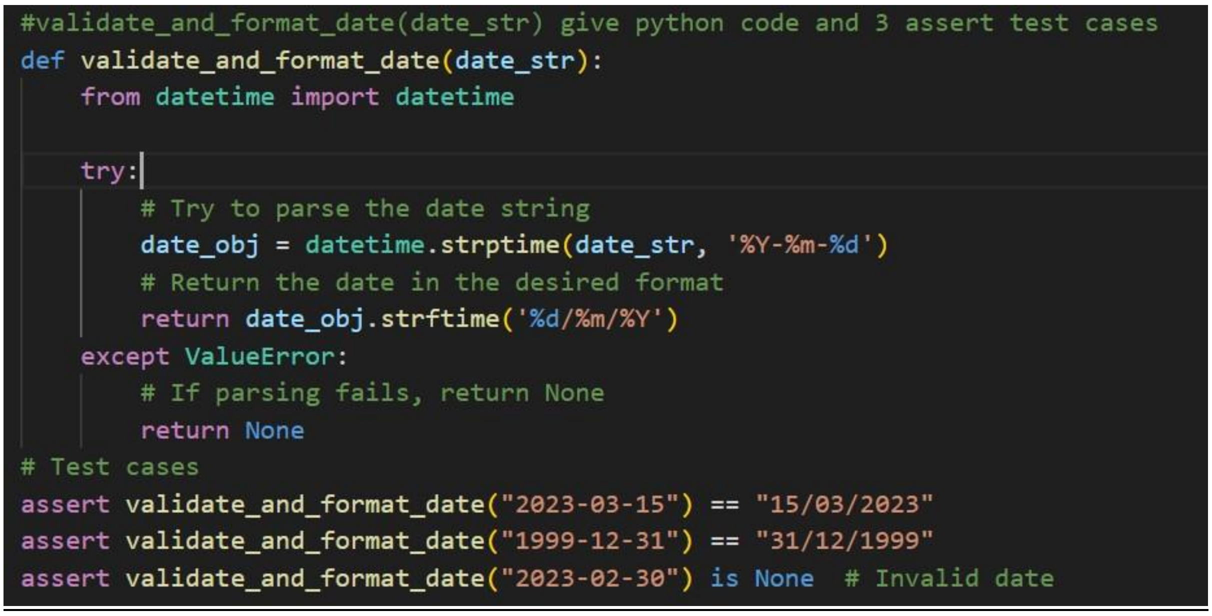
Output #5:

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

<u>Prompt:</u>

#validate_and_format_date(date_str) give python code and 3 assert test cases **Given**

Code:



```python
#validate_and_format_date(date_str) give python code and 3 assert test cases
def validate_and_format_date(date_str):
    from datetime import datetime

    try:
        # Try to parse the date string
        date_obj = datetime.strptime(date_str, '%Y-%m-%d')
        # Return the date in the desired format
        return date_obj.strftime('%d/%m/%Y')
    except ValueError:
        # If parsing fails, return None
        return None
# Test cases
assert validate_and_format_date("2023-03-15") == "15/03/2023"
assert validate_and_format_date("1999-12-31") == "31/12/1999"
assert validate_and_format_date("2023-02-30") is None  # Invalid date
```