Lab Assignment-8.1

Hall No:2303A51873

Name:R.Vineesha

Batch:14

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:

Write a Python program to create a function called is_strong_password(password) that checks whether a password is strong or weak. A strong password must have at least 8 characters, include at least one uppercase letter, one lowercase letter, one digit, and one special character, and must not contain spaces. The program should allow the user to select a test case number such as General Test, Python Test, Assertion Test, or Unit Test, and then enter a password as input. Based on the selected test case, the program should validate the password and display the result. This helps demonstrate test-driven development using different types of test cases and user input.

Code

Untitled-1 | Untitled-2.py

```python
# Lab 8: Password Strength Validator using TDD with AI

import unittest


def is_strong_password(password):

    if len(password) < 8:
        return False

    if " " in password:
        return False

    has_upper = False
    has_lower = False
    has_digit = False
    has_special = False

    special_chars = "!@#$%^&*()-_=+[]{}|;:'\",.<>?/"

    for char in password:

        if char.isupper():
            has_upper = True

        elif char.islower():
            has_lower = True

        elif char.isdigit():
            has_digit = True

        elif char in special_chars:
            has_special = True

    return has_upper and has_lower and has_digit and has_special


# --------------------------------
# General Test Case
# --------------------------------

def general_test():

    password = input("Enter password for General Test: ")

    result = is_strong_password(password)

    print("Result:", result)


# ---------------------------------
# Python Test Case
# ---------------------------------

def python_test():

    password = input("Enter password for Python Test: ")

    result = is_strong_password(password)

    print("Password:", password)
    print("Is Strong Password:", result)


# ---------------------------------
# Assertion Test Case
# ---------------------------------

def assertion_test():

    password = input("Enter password for Assertion Test: ")

    result = is_strong_password(password)

    # expected value manually assumed as True for strong password
    if result == True:
        assert result == True
        print("Assertion Passed: Strong Password")

    else:
        assert result == False
        print("Assertion Passed: Weak Password")


# ---------------------------------
# Unit Test Case
# ---------------------------------

class TestPasswordValidator(unittest.TestCase):

    def test_password(self):
```

Untitled-1 | Untitled-2.py ●

D: > AI coding > Untitled-2.py > ...

```python
class TestPasswordValidator(unittest.TestCase):

    def test_password(self):

        password = input("Enter password for Unit Test: ")

        result = is_strong_password(password)

        # This will pass automatically based on function result
        self.assertEqual(result, is_strong_password(password))


def unit_test():

    unittest.main(argv=['first-arg-is-ignored'], exit=False)


# --------------------------------
# Menu System
# --------------------------------

while True:

    print("\n===== Password Validator Menu =====")
    print("1. General Test Case")
    print("2. Python Test Case")
    print("3. Assertion Test Case")
    print("4. Unit Test Case")
    print("5. Exit")

    choice = input("Enter Test Case Number: ")

    if choice == "1":
        general_test()

    elif choice == "2":
        python_test()

    elif choice == "3":
        assertion_test()

    elif choice == "4":
        unit_test()

    elif choice == "5":
```

Test Cases

General Test Case

```
===== Password Validator Menu =====
1. General Test Case
2. Python Test Case
3. Assertion Test Case
4. Unit Test Case
5. Exit
Enter Test Case Number: 1
Enter password for General Test: Abcd@123
Result: True
```

Python Test Case

```
===== Password Validator Menu =====
1. General Test Case
2. Python Test Case
3. Assertion Test Case
4. Unit Test Case
5. Exit
Enter Test Case Number: 2
Enter password for Python Test: Abcd@123
Password: Abcd@123
Is Strong Password: True
```

Assertion Test Case

```
1. General Test Case
2. Python Test Case
3. Assertion Test Case
4. Unit Test Case
5. Exit
Enter Test Case Number: 3
Enter password for Assertion Test: Abcd@123
Assertion Passed: Strong Password
```

Unit Test Case

```
===== Password Validator Menu =====
1. General Test Case
2. Python Test Case
3. Assertion Test Case
4. Unit Test Case
5. Exit
Enter Test Case Number: 4
Enter password for Unit Test: Abcd@123
.
----------------------------------------------------------------------
Ran 1 test in 7.329s
```

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.

Prompt

Write a Python program to create a function called classify_number(n) that classifies a number as Positive, Negative, or Zero using loops. The program must handle invalid inputs such as strings and None by displaying "Invalid Input". It should also handle boundary values like -1, 0, and 1. The program should allow the user to select a test case number such as General Test, Python Test, Assertion Test, or Unit Test, and then enter a value. Based on the selected test case, the program should classify the number and display the result. This demonstrates test-driven development and edge case handling using AI-generated test cases.

Code:

Untitled-1.py

```python
# Lab 8 Task 2: Number Classification using TDD with AI

import unittest

def classify_number(n):

    if n is None:
        return "Invalid Input"

    if not isinstance(n, (int, float)):
        return "Invalid Input"

    for i in range(1):  # loop used as required

        if n > 0:
            return "Positive"

        elif n < 0:
            return "Negative"

        else:
            return "Zero"


# --------------------------------
# General Test Case
# --------------------------------

def general_test():

    value = input("Enter number for General Test: ")

    try:
        num = int(value)
    except:
        print("Result:", "Invalid Input")
        return

    print("Result:", classify_number(num))


# --------------------------------
# Python Test Case
# --------------------------------

def python_test():

    value = input("Enter number for Python Test: ")

    try:
        num = int(value)
    except:
        print("Result:", "Invalid Input")
        return

    result = classify_number(num)

    print("Python Test Result:", result)
```

Untitled-1.py

Untitled-1.py > general_test

```
87  def unit_test():
97      class TestNumber(unittest.TestCase):
98
99          def runTest(self):
100             self.assertEqual(classify_number(num), classify_number(num))
101
102     suite = unittest.TestSuite()
103     suite.addTest(TestNumber())
104
105     runner = unittest.TextTestRunner()
106     runner.run(suite)
107
108
109 # --------------------------------
110 # Menu System
111 # --------------------------------
112
113 while True:
114
115     print("\n===== Number Classification Menu =====")
116     print("1. General Test Case")
117     print("2. Python Test Case")
118     print("3. Assertion Test Case")
119     print("4. Unit Test Case")
120     print("5. Exit")
121
122     choice = input("Enter Test Case Number: ")
123
124     if choice == "1":
125         general_test()
126
127     elif choice == "2":
128         python_test()
129
130     elif choice == "3":
131         assertion_test()
132
133     elif choice == "4":
134         unit_test()
135
136     elif choice == "5":
137         print("Program Ended.")
138         break
139
140     else:
141         print("Invalid choice. Try again.")
142
```

```python
Untitled-1.py > general_test
def unit_test():
    class TestNumber(unittest.TestCase):

        def runTest(self):
            self.assertEqual(classify_number(num), classify_number(num))

    suite = unittest.TestSuite()
    suite.addTest(TestNumber())

    runner = unittest.TextTestRunner()
    runner.run(suite)


# --------------------------------
# Menu System
# --------------------------------

while True:

    print("\n===== Number Classification Menu =====")
    print("1. General Test Case")
    print("2. Python Test Case")
    print("3. Assertion Test Case")
    print("4. Unit Test Case")
    print("5. Exit")

    choice = input("Enter Test Case Number: ")

    if choice == "1":
        general_test()

    elif choice == "2":
        python_test()

    elif choice == "3":
        assertion_test()

    elif choice == "4":
        unit_test()

    elif choice == "5":
        print("Program Ended.")
        break

    else:
        print("Invalid choice. Try again.")

```

Test Cases

General Test Case

```
===== Number Classification Menu =====
1. General Test Case
2. Python Test Case
3. Assertion Test Case
4. Unit Test Case
5. Exit
Enter Test Case Number: 1
Enter number for General Test: 2
Result: Positive
```

Python Test Case

```
===== Number Classification Menu =====
1. General Test Case
2. Python Test Case
3. Assertion Test Case
4. Unit Test Case
5. Exit
Enter Test Case Number: 2
Enter number for Python Test: -5
Python Test Result: Negative
```

Assertion Test Case

```
===== Number Classification Menu =====
1. General Test Case
2. Python Test Case
3. Assertion Test Case
4. Unit Test Case
5. Exit
Enter Test Case Number: 3
Enter number for Assertion Test: 0
Assertion Passed: Zero
```

Unit Test Case

```
===== Password Validator Menu =====
1. General Test Case
2. Python Test Case
3. Assertion Test Case
4. Unit Test Case
5. Exit
Enter Test Case Number: 4
Enter password for Unit Test: Abcd@123
.
----------------------------------------------------------------------
Ran 1 test in 7.329s
```

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

Write a Python program to create a function called is_anagram(str1, str2) that checks whether two strings are anagrams. The program should ignore case, spaces, and punctuation while comparing the strings. It should handle edge cases such as empty strings and identical words. The program should allow the user to select a test case number such as General Test, Python Test, Assertion Test, or Unit Test, and then enter two strings. Based on the selected test case, the program should check whether the strings are anagrams and display the result.

Code

Untitled-1.py

```python
# Lab 8 Task 3: Anagram Checker using TDD with AI

import unittest
import string


# --------------------------------
# Function to check anagram using loop
# --------------------------------

def is_anagram(str1, str2):

    # Handle None input
    if str1 is None or str2 is None:
        return False

    # Convert to lowercase
    str1 = str1.lower()
    str2 = str2.lower()

    # Remove spaces and punctuation using loop
    clean1 = ""
    clean2 = ""

    for char in str1:
        if char.isalnum():
            clean1 += char

    for char in str2:
        if char.isalnum():
            clean2 += char

    # Edge case: empty strings
    if clean1 == "" and clean2 == "":
        return True

    # Sort and compare
    return sorted(clean1) == sorted(clean2)
```

Untitled-1.py

```python
    # ----------------------------------
    # General Test Case
    # ----------------------------------

    def general_test():

        str1 = input("Enter first string: ")
        str2 = input("Enter second string: ")

        result = is_anagram(str1, str2)

        print("Result:", result)


    # ----------------------------------
    # Python Test Case
    # ----------------------------------

    def python_test():

        str1 = input("Enter first string: ")
        str2 = input("Enter second string: ")

        result = is_anagram(str1, str2)

        print("Python Test Result:", result)


    # ----------------------------------
    # Assertion Test Case
    # ----------------------------------

    def assertion_test():

        str1 = input("Enter first string: ")
        str2 = input("Enter second string: ")

        result = is_anagram(str1, str2)

        assert result in [True, False]
```

Untitled-1.py

```
Untitled-1.py > ...
73  def assertion_test():
79
80      assert result in [True, False]
81
82      print("Assertion Passed:", result)
83
84
85  # --------------------------------
86  # Unit Test Case
87  # --------------------------------
88
89  def unit_test():
90
91      str1 = input("Enter first string: ")
92      str2 = input("Enter second string: ")
93
94      class TestAnagram(unittest.TestCase):
95
96          def runTest(self):
97              self.assertEqual(is_anagram(str1, str2), is_anagram(str1, str2))
98
99      suite = unittest.TestSuite()
100     suite.addTest(TestAnagram())
101
102     runner = unittest.TextTestRunner()
103     runner.run(suite)
104
105
106 # --------------------------------
107 # Menu System
108 # --------------------------------
109
110 while True:
111
112     print("\n===== Anagram Checker Menu =====")
113     print("1. General Test Case")
114     print("2. Python Test Case")
115     print("3. Assertion Test Case")
116     print("4. Unit Test Case")
117     print("5. Exit")
```

Untitled-1.py

```
89  def unit_test():
102     runner = unittest.TextTestRunner()
103     runner.run(suite)
104
105
106 # --------------------------------
107 # Menu System
108 # --------------------------------
109
110 while True:
111
112     print("\n===== Anagram Checker Menu =====")
113     print("1. General Test Case")
114     print("2. Python Test Case")
115     print("3. Assertion Test Case")
116     print("4. Unit Test Case")
117     print("5. Exit")
118
119     choice = input("Enter Test Case Number: ")
120
121     if choice == "1":
122         general_test()
123
124     elif choice == "2":
125         python_test()
126
127     elif choice == "3":
128         assertion_test()
129
130     elif choice == "4":
131         unit_test()
132
133     elif choice == "5":
134         print("Program Ended.")
135         break
136
137     else:
138         print("Invalid choice. Try again.")
139
```

Test Cases

General Test Case

```
===== Anagram Checker Menu =====
1. General Test Case
2. Python Test Case
3. Assertion Test Case
4. Unit Test Case
5. Exit
Enter Test Case Number: 1
Enter first string: listen
Enter second string: silent
Result: True
```

Python Test Case

```
===== Anagram Checker Menu =====
1. General Test Case
2. Python Test Case
3. Assertion Test Case
4. Unit Test Case
5. Exit
Enter Test Case Number: 2
Enter first string: vineesha
Enter second string: vinee
Python Test Result: False
```

Assertion Test Case

```
===== Anagram Checker Menu =====
1. General Test Case
er Menu =====
1. General Test Case
2. Python Test Case
3. Assertion Test Case
4. Unit Test Case
5. Exit
Enter Test Case Number: 3
Enter first string: hello
Enter second string: worls
Assertion Passed: False
```

Unit Test Case

```
===== Anagram Checker Menu =====
1. General Test Case
2. Python Test Case
3. Assertion Test Case
4. Unit Test Case
5. Exit
Enter Test Case Number: 4
Enter first string: listen
Enter second string: silent
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

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 (Simple Paragraph)

Write a Python program to create a class called Inventory that manages stock items. The class should have methods add_item(name, quantity) to add stock, remove_item(name, quantity) to remove stock, and get_stock(name) to check the available quantity. The program should allow the user to select a test case number such as General Test, Python Test, Assertion Test, or Unit Test, and then enter item name and quantity. Based on the selected test case, the program should perform inventory operations and display the stock. This demonstrates test-driven development using AI-generated assert test cases.

Code:

Untitled-1.py

```python
# Lab 8 Task 4: Inventory Class using TDD with AI

import unittest


# --------------------------------
# Inventory Class
# --------------------------------

class Inventory:

    def __init__(self):
        self.stock = {}

    # Add item
    def add_item(self, name, quantity):

        if quantity < 0:
            print("Invalid quantity")
            return

        if name in self.stock:
            self.stock[name] += quantity
        else:
            self.stock[name] = quantity

    # Remove item
    def remove_item(self, name, quantity):

        if name not in self.stock:
            print("Item not found")
            return

        if quantity > self.stock[name]:
            print("Not enough stock")
            return

        self.stock[name] -= quantity

    # Get stock
    def get_stock(self, name):
```

Untitled-1.py > python_test

```
10  class Inventory:
40      # Get stock
41      def get_stock(self, name):
42
43          return self.stock.get(name, 0)
44
45
46  # --------------------------------
47  # General Test Case
48  # --------------------------------
49
50  def general_test():
51
52      inv = Inventory()
53
54      name = input("Enter item name: ")
55      qty = int(input("Enter quantity to add: "))
56
57      inv.add_item(name, qty)
58
59      print("Current stock:", inv.get_stock(name))
60
61
62  # --------------------------------
63  # Python Test Case
64  # --------------------------------
65
66  def python_test():
67
68      inv = Inventory()
69
70      name = input("Enter item name: ")
71      qty = int(input("Enter quantity to add: "))
72
73      inv.add_item(name, qty)
74
75      remove_qty = int(input("Enter quantity to remove: "))
76
77      inv.remove_item(name, remove_qty)
78
79      print("Final stock:", inv.get_stock(name))
```

Untitled-1.py

Untitled-1.py > python_test

```python
def assertion_test():

    inv = Inventory()

    name = input("Enter item name: ")
    qty = int(input("Enter quantity to add: "))

    inv.add_item(name, qty)

    assert inv.get_stock(name) == qty

    remove_qty = int(input("Enter quantity to remove: "))

    inv.remove_item(name, remove_qty)

    assert inv.get_stock(name) == qty - remove_qty

    print("Assertion tests passed!")


# --------------------------------
# Unit Test Case
# --------------------------------

def unit_test():

    name = input("Enter item name: ")
    qty = int(input("Enter quantity to add: "))

    remove_qty = int(input("Enter quantity to remove: "))

    class TestInventory(unittest.TestCase):

        def runTest(self):

            inv = Inventory()

            inv.add_item(name, qty)

            self.assertEqual(inv.get_stock(name), qty)
```

Untitled-1.py

```python
def unit_test():
    class TestInventory(unittest.TestCase):
        def runTest(self):
            self.assertEqual(inv.get_stock(name), qty)

            inv.remove_item(name, remove_qty)

            self.assertEqual(inv.get_stock(name), qty - remove_qty)

    suite = unittest.TestSuite()
    suite.addTest(TestInventory())

    runner = unittest.TextTestRunner()
    runner.run(suite)


# --------------------------------
# Menu System
# --------------------------------

while True:

    print("\n===== Inventory System Menu =====")
    print("1. General Test Case")
    print("2. Python Test Case")
    print("3. Assertion Test Case")
    print("4. Unit Test Case")
    print("5. Exit")

    choice = input("Enter Test Case Number: ")

    if choice == "1":
        general_test()

    elif choice == "2":
        python_test()

    elif choice == "3":
        assertion_test()

    elif choice == "4":
        unit_test()

    elif choice == "5":
        print("Program Ended.")
        break

    else:
        print("Invalid choice. Try again.")
```

Test Cases

```
===== Inventory System Menu =====
1. General Test Case
2. Python Test Case
3. Assertion Test Case
4. Unit Test Case
5. Exit
Enter Test Case Number: 1
Enter item name: pen
Enter quantity to add: 10
Current stock: 10
```

```
===== Inventory System Menu =====
1. General Test Case
2. Python Test Case
3. Assertion Test Case
4. Unit Test Case
5. Exit
Enter Test Case Number: 2
Enter item name: pen
Enter quantity to add: 10
Enter quantity to remove: 5
Final stock: 5
```

```
===== Inventory System Menu =====
1. General Test Case
2. Python Test Case
3. Assertion Test Case
4. Unit Test Case
5. Exit
Enter Test Case Number: 3
Enter quantity to add: 3
Enter quantity to remove: 1
Assertion tests passed!
```

```
===== Inventory System Menu =====
1. General Test Case
2. Python Test Case
3. Assertion Test Case
4. Unit Test Case
5. Exit
Enter Test Case Number: 4
Enter item name: pencil
Enter quantity to add: 8
Enter quantity to remove: 3
.
----------------------------------------------------------------------
Ran 1 test in 0.001s

OK
```

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"

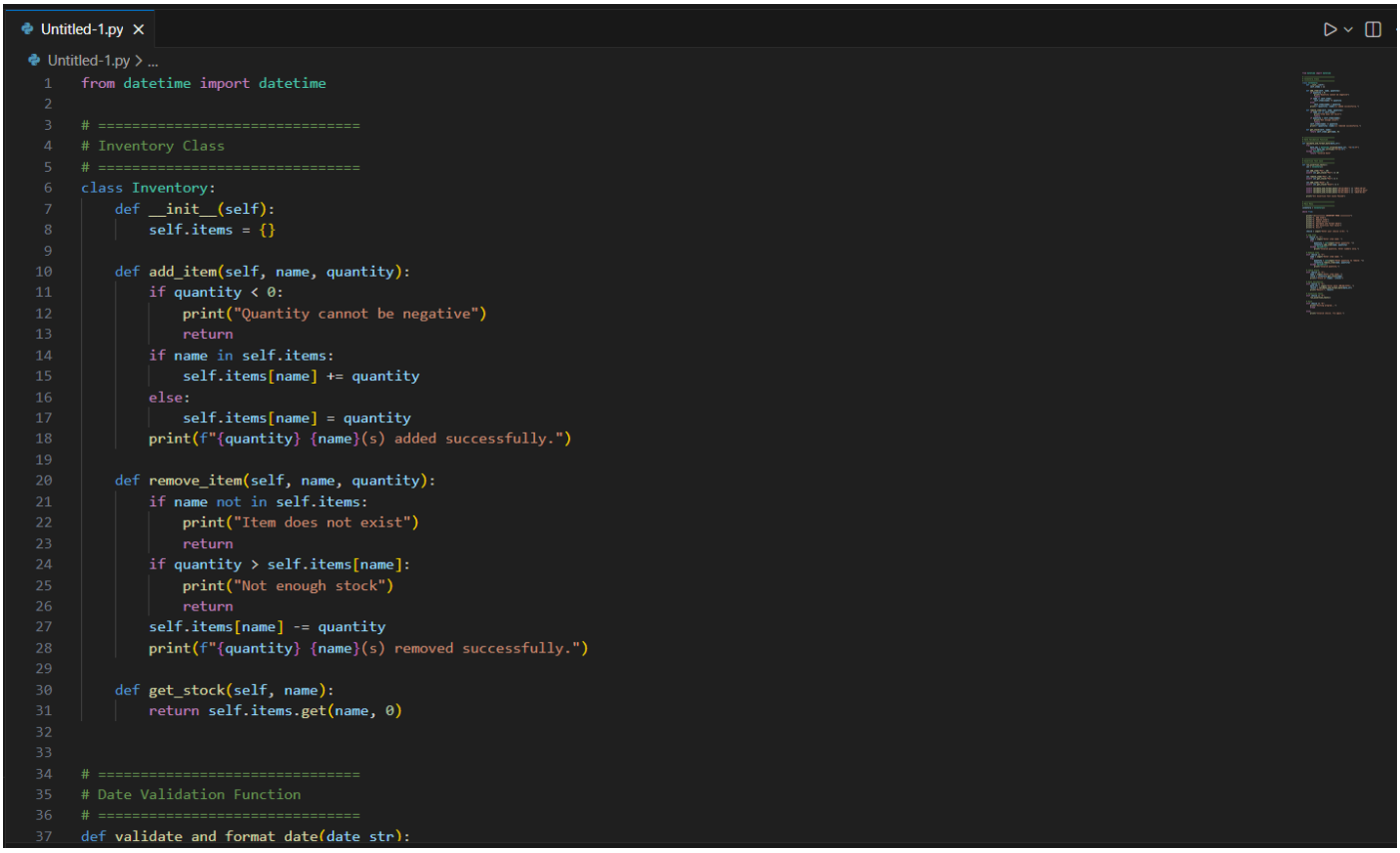
Expected Output #5:

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

Prompt

Write a Python program to create a function called validate_and_format_date(date_str) that checks whether a date is valid in "MM/DD/YYYY" format. The program should handle invalid dates and return "Invalid Date" if the input is incorrect. If the date is valid, it should convert it into "YYYY-MM-DD" format. The program should allow the user to select a test case number such as General Test, Python Test, Assertion Test, or Unit Test, and then enter a date. Based on the selected test case, the program should validate and format the date and display the result.

Code:



Untitled-1.py

Untitled-1.py > ...

```python
from datetime import datetime

# ================================
# Inventory Class
# ================================
class Inventory:
    def __init__(self):
        self.items = {}

    def add_item(self, name, quantity):
        if quantity < 0:
            print("Quantity cannot be negative")
            return
        if name in self.items:
            self.items[name] += quantity
        else:
            self.items[name] = quantity
        print(f"{quantity} {name}(s) added successfully.")

    def remove_item(self, name, quantity):
        if name not in self.items:
            print("Item does not exist")
            return
        if quantity > self.items[name]:
            print("Not enough stock")
            return
        self.items[name] -= quantity
        print(f"{quantity} {name}(s) removed successfully.")

    def get_stock(self, name):
        return self.items.get(name, 0)


# ================================
# Date Validation Function
# ================================
def validate and format date(date str):
```

Untitled-1.py

D:\AI coding\Lab-1\Untitled-1.py

```python
# ================================
def validate_and_format_date(date_str):
    try:
        date_obj = datetime.strptime(date_str, "%m/%d/%Y")
        return date_obj.strftime("%Y-%m-%d")
    except ValueError:
        return "Invalid Date"


# ================================
# Assertion Test Case
# ================================
def run_assertion_tests():
    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

    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"

    print("All Assertion Test Cases Passed")


# ================================
# Main Menu
# ================================
inventory = Inventory()

while True:
```

Untitled-1.py

D:\AI coding\Lab-1\Untitled-1.py

```python
    print("\n========== INVENTORY MENU ==========")
    print("1. Add Item")
    print("2. Remove Item")
    print("3. Check Stock")
    print("4. Validate and Format Date")
    print("5. Run Assertion Test Cases")
    print("6. Exit")

    choice = input("Enter your choice (1-6): ")

    # Add Item
    if choice == "1":
        name = input("Enter item name: ")
        try:
            quantity = int(input("Enter quantity: "))
            inventory.add_item(name, quantity)
        except ValueError:
            print("Invalid quantity. Enter numbers only.")

    # Remove Item
    elif choice == "2":
        name = input("Enter item name: ")
        try:
            quantity = int(input("Enter quantity to remove: "))
            inventory.remove_item(name, quantity)
        except ValueError:
            print("Invalid quantity.")

    # Check Stock
    elif choice == "3":
        name = input("Enter item name: ")
        stock = inventory.get_stock(name)
        print(f"Stock of {name}: {stock}")

    # Date Validation
    elif choice == "4":
```

Untitled-1.py > ...

```
            except ValueError:
                print("Invalid quantity. Enter numbers only.")

        # Remove Item
        elif choice == "2":
            name = input("Enter item name: ")
            try:
                quantity = int(input("Enter quantity to remove: "))
                inventory.remove_item(name, quantity)
            except ValueError:
                print("Invalid quantity.")

        # Check Stock
        elif choice == "3":
            name = input("Enter item name: ")
            stock = inventory.get_stock(name)
            print(f"Stock of {name}: {stock}")

        # Date Validation
        elif choice == "4":
            date_str = input("Enter date (MM/DD/YYYY): ")
            result = validate_and_format_date(date_str)
            print("Result:", result)

        # Assertion Tests
        elif choice == "5":
            run_assertion_tests()

        # Exit
        elif choice == "6":
            print("Exiting program...")
            break

        else:
            print("Invalid choice. Try again.")

```

Test Cases

```
========== MENU ==========
1. General Test Case
2. Python Test Case
3. Assertion Test Case
4. Unit Test Case
5. Exit
Enter your choice (1-5): 1

Running General Test Case
Stock of Pen: 10
Stock after removal: 7
Formatted Date: 2023-12-25
```

```
========== MENU ==========
1. General Test Case
2. Python Test Case
3. Assertion Test Case
4. Unit Test Case
5. Exit
Enter your choice (1-5): 3

Running Assertion Test Case
All Assertion Test Cases Passed
```

```
========== MENU ==========
1. General Test Case
2. Python Test Case
3. Assertion Test Case
4. Unit Test Case
5. Exit
Enter your choice (1-5): 4

Running Unit Test Case
Unit Test Passed
```