AI ASSISTED CODING

ASSIGNMENT-8.1

Name:MD ZIAUDDIN

Rollno:2403A51271

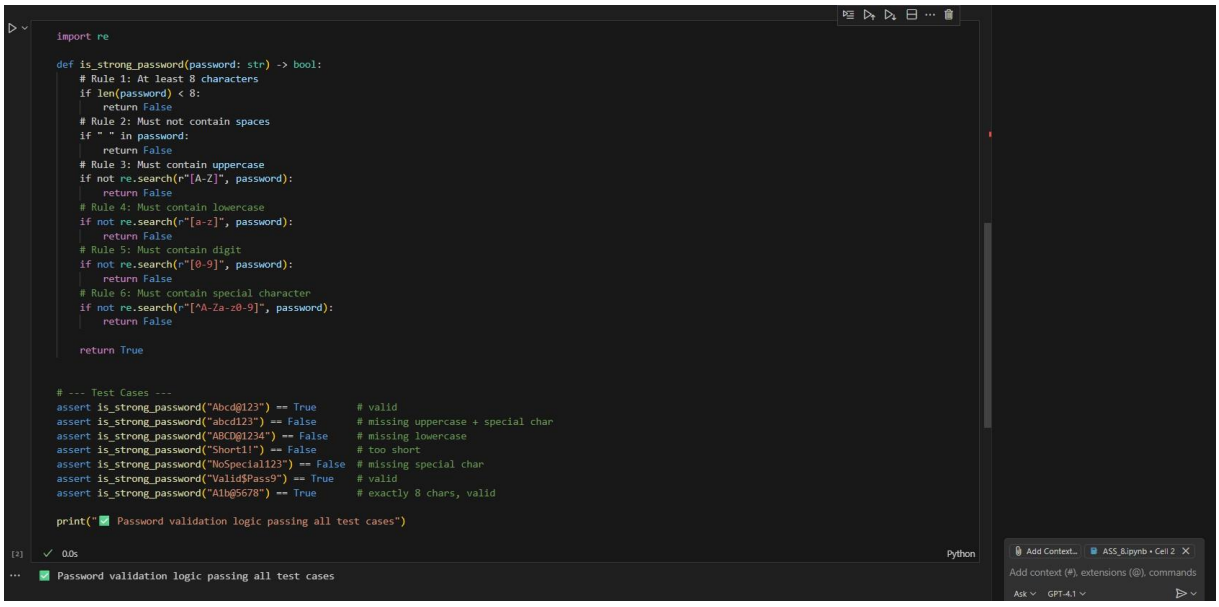
Task-1:

Prompt:

generate at least 3 assert test cases foris_strong_password(password) and implement the validator
function and Password must have at least 8 characters also include uppercase, lowercase, digit, and special character,must not contain spaces. assert test cases examples like assert
is_strong_password("Abcd@123") == True
assert is_strong_password("abcd123") == False
assert is_strong_password("ABCD@1234") == True

finally,must be like Password validation logic passing allgenerated test cases

Code with output:



```
import re

def is_strong_password(password: str) -> bool:
    # Rule 1: At least 8 characters
    if len(password) < 8:
        return False
    # Rule 2: Must not contain spaces
    if " " in password:
        return False
    # Rule 3: Must contain uppercase
    if not re.search(r"[A-Z]", password):
        return False
    # Rule 4: Must contain lowercase
    if not re.search(r"[a-z]", password):
        return False
    # Rule 5: Must contain digit
    if not re.search(r"[0-9]", password):
        return False
    # Rule 6: Must contain special character
    if not re.search(r"[^A-Za-z0-9]", password):
        return False

    return True


# --- Test Cases ---
assert is_strong_password("Abcd@123") == True      # valid
assert is_strong_password("abcd123") == False      # missing uppercase + special char
assert is_strong_password("ABCD@1234") == False    # missing lowercase
assert is_strong_password("Short1!") == False      # too short
assert is_strong_password("NoSpecial123") == False  # missing special char
assert is_strong_password("Valid$Pass9") == True   # valid
assert is_strong_password("A1b@5678") == True      # exactly 8 chars, valid

print("✅ Password validation logic passing all test cases")
```

```
[2]  ✓ 0.0s                                                    Python
...  ✅ Password validation logic passing all test cases
```

Task-2:

Prompt:

generate at least 3 assert test cases for a classify_number(n) function. Implement using loops.

The requirements are Classify numbers as Positive, Negative, or Zero.,Handle invalid inputs like strings and None., 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"
output should be like Classification logic passing all assert tests

Code with output:

```python
def classify_number(n):
    # Loop used to validate input (simple simulation)
    for _ in range(1):  # single iteration just to demonstrate loop usage
        if n is None:
            return "Invalid Input"
        if isinstance(n, (int, float)):
            if n > 0:
                return "Positive"
            elif n < 0:
                return "Negative"
            else:
                return "Zero"
        else:
            return "Invalid Input"


# --- Test Cases ---
assert classify_number(10) == "Positive"    # normal positive
assert classify_number(-5) == "Negative"    # normal negative
assert classify_number(0) == "Zero"         # boundary zero
assert classify_number(-1) == "Negative"    # boundary condition
assert classify_number(1) == "Positive"     # boundary condition
assert classify_number("abc") == "Invalid Input"  # invalid string
assert classify_number(None) == "Invalid Input"   # invalid None

print("✅ Classification logic passing all assert tests")
```

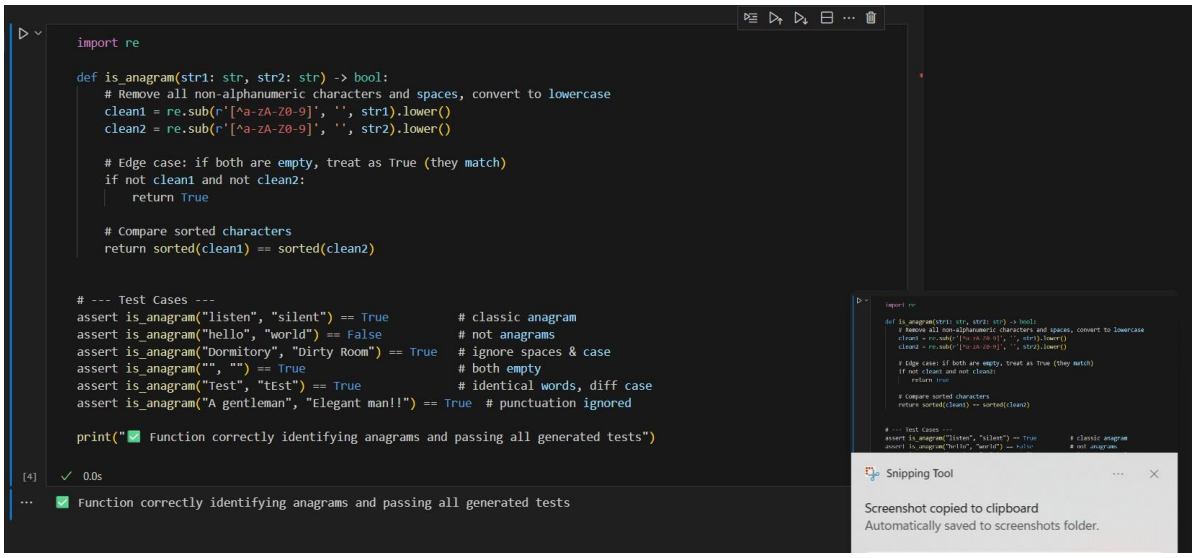
```
[3]  ✓ 0.0s                                                    Python
...  ✅ Classification logic passing all assert tests
```

Task-3:

Prompt:

generate at least 3 assert test cases for is_anagram(str1, str2) and implement the function.
the Requirements are like Ignore case, spaces, and punctuation,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
Output should like a Function correctly identifying anagrams and passing all
generated tests

Code with output:

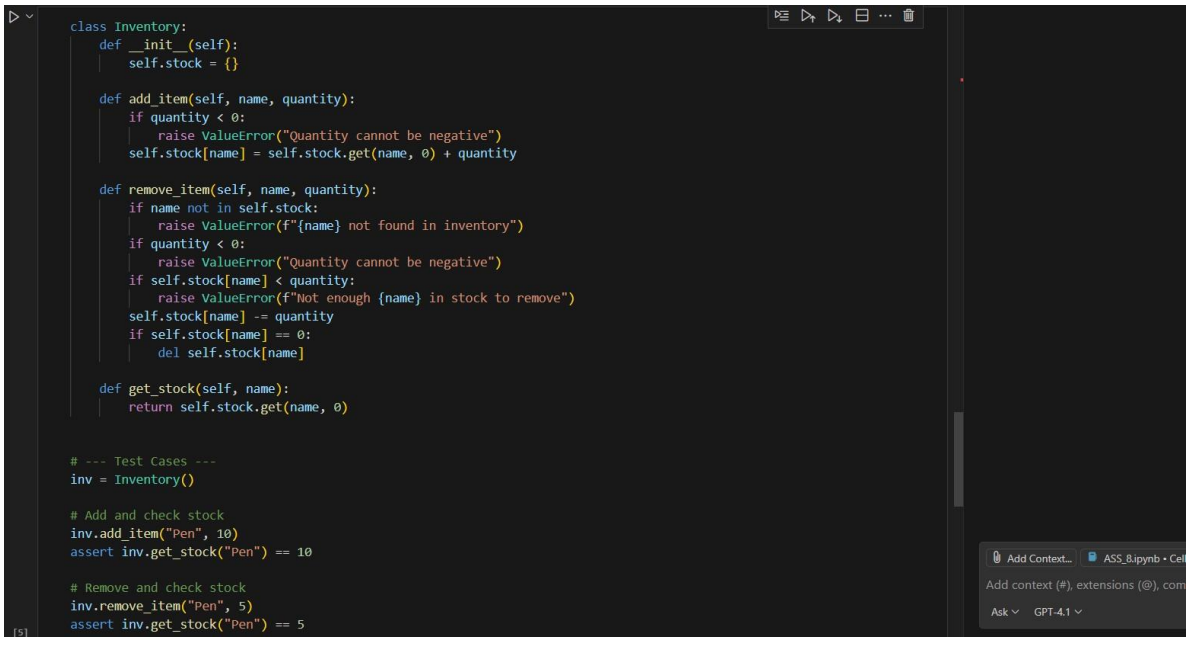


Task-4:

Prompt:

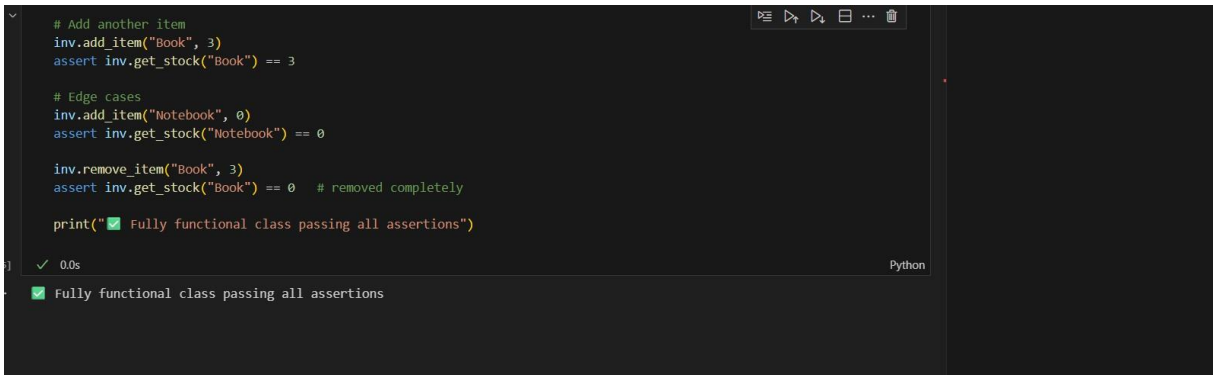
generate at least 3 assert-based tests for an Inventory class with stock management.
Methods:
o add_item(name, quantity)
o remove_item(name, quantity)
o get_stock(name)
Example Assert Test Cases:
inv = Inventory()
inv.add_item("Pen", 10)
assert inv.get_stock("Pen") == 10
inv.remove_item("Pen", 5)
assert inv.get_stock("Pen") == 5
inv.add_item("Book", 3)
assert inv.get_stock("Book") == 3
Output like Fully functional class passing all assertions

Code with output:



```
class Inventory:
    def __init__(self):
        self.stock = {}

    def add_item(self, name, quantity):
        if quantity < 0:
            raise ValueError("Quantity cannot be negative")
        self.stock[name] = self.stock.get(name, 0) + quantity

    def remove_item(self, name, quantity):
        if name not in self.stock:
            raise ValueError(f"{name} not found in inventory")
        if quantity < 0:
            raise ValueError("Quantity cannot be negative")
        if self.stock[name] < quantity:
            raise ValueError(f"Not enough {name} in stock to remove")
        self.stock[name] -= quantity
        if self.stock[name] == 0:
            del self.stock[name]

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


# --- Test Cases ---
inv = Inventory()

# Add and check stock
inv.add_item("Pen", 10)
assert inv.get_stock("Pen") == 10

# Remove and check stock
inv.remove_item("Pen", 5)
assert inv.get_stock("Pen") == 5
```

Add Context... ASS_8.ipynb • Cell
Add context (#), extensions (@), com
Ask GPT-4.1



Task-5:

Prompt:

generate at least 3 assert test cases for validate_and_format_date(date_str) to check and convert dates. The 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"
Output like a Function passes all generated assertions and handles edge
cases

Code with output:

```python
from datetime import datetime

def validate_and_format_date(date_str: str) -> str:
    try:
        # Parse with expected MM/DD/YYYY format
        dt = datetime.strptime(date_str, "%m/%d/%Y")
        return dt.strftime("%Y-%m-%d")
    except ValueError:
        return "Invalid Date"


# --- Test Cases ---
assert validate_and_format_date("10/15/2023") == "2023-10-15"  # valid
assert validate_and_format_date("02/30/2023") == "Invalid Date" # invalid day
assert validate_and_format_date("01/01/2024") == "2024-01-01"  # valid
assert validate_and_format_date("13/01/2023") == "Invalid Date" # invalid month
assert validate_and_format_date("00/10/2023") == "Invalid Date" # invalid month
assert validate_and_format_date("12/31/1999") == "1999-12-31"  # valid edge case

print("✅ Function passes all generated assertions and handles edge cases")
```

(function) def validate_and_format_date(date_str: str) -> str

[6] ✓ 0.0s Py

✅ Function passes all generated assertions and handles edge cases