Lab Assignment 8.1

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:

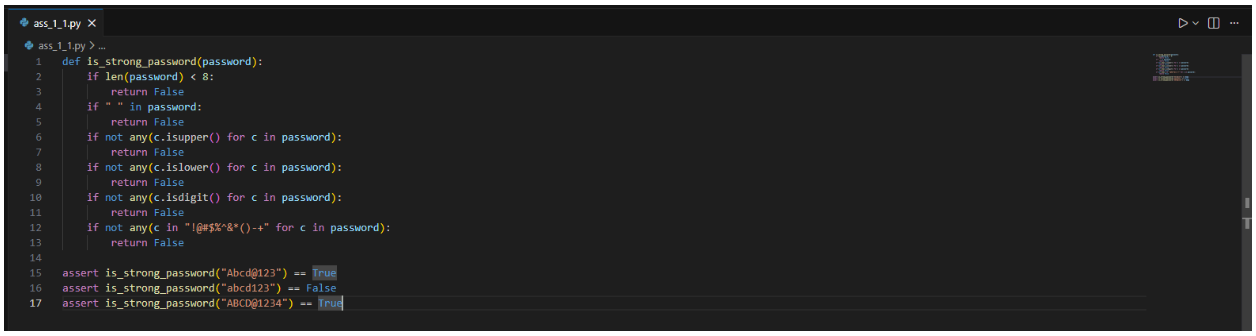
- Passwords must have at least 8 characters.
- Must include uppercase, lowercase, digit, and special character.
- Must not contain spaces.

Example Assert Test Cases:

- assert is_strong_password("Abcd@123") == True
- assert is_strong_password("abcd123") == False
- assert is_strong_password("ABCD@1234") == True

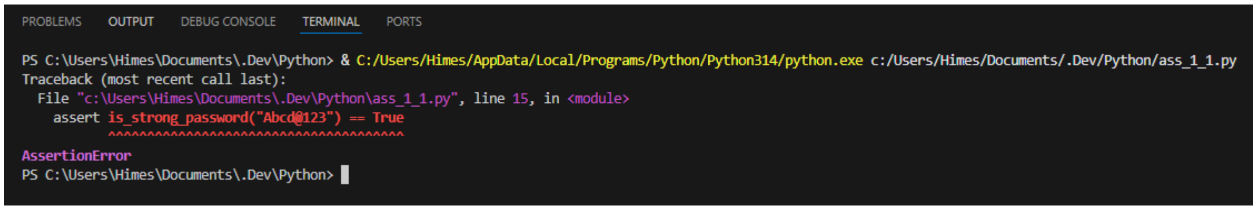
Expected Output #1:

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



```python
def is_strong_password(password):
    if len(password) < 8:
        return False
    if " " in password:
        return False
    if not any(c.isupper() for c in password):
        return False
    if not any(c.islower() for c in password):
        return False
    if not any(c.isdigit() for c in password):
        return False
    if not any(c in "!@#$%^&*()-+" for c in password):
        return False

assert is_strong_password("Abcd@123") == True
assert is_strong_password("abcd123") == False
assert is_strong_password("ABCD@1234") == True
```

Output:



```
PS C:\Users\Himes\Documents\.Dev\Python> & C:/Users/Himes/AppData/Local/Programs/Python/Python314/python.exe c:/Users/Himes/Documents/.Dev/Python/ass_1_1.py
Traceback (most recent call last):
  File "c:\Users\Himes\Documents\.Dev\Python\ass_1_1.py", line 15, in <module>
    assert is_strong_password("Abcd@123") == True
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

AssertionError
PS C:\Users\Himes\Documents\.Dev\Python>
```

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:

- Classify numbers as Positive, Negative, or Zero.
- Handle invalid inputs like strings and None.
- Include boundary conditions (-1, 0, 1).

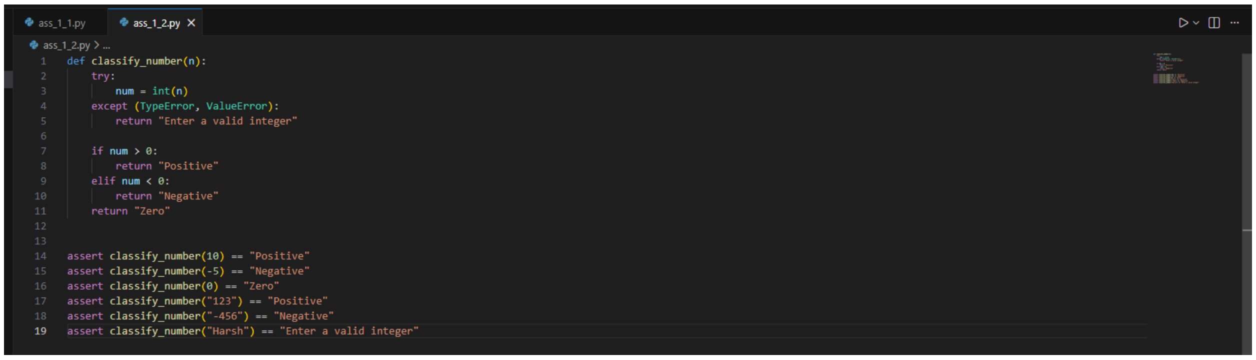
Example Assert Test Cases:

- assert classify_number(10) == "Positive"
- assert classify_number(-5) == "Negative"

- assert classify_number(0) == "Zero"

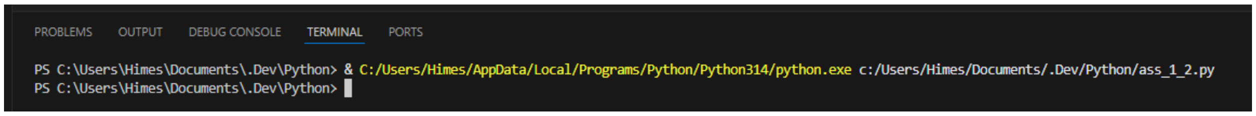
Expected Output:

Classification logic passing all assert tests.



```python
def classify_number(n):
    try:
        num = int(n)
    except (TypeError, ValueError):
        return "Enter a valid integer"

    if num > 0:
        return "Positive"
    elif num < 0:
        return "Negative"
    return "Zero"


assert classify_number(10) == "Positive"
assert classify_number(-5) == "Negative"
assert classify_number(0) == "Zero"
assert classify_number("123") == "Positive"
assert classify_number("-456") == "Negative"
assert classify_number("Harsh") == "Enter a valid integer"
```

Output:



```
PS C:\Users\Himes\Documents\.Dev\Python> & C:/Users/Himes/AppData/Local/Programs/Python/Python314/python.exe c:/Users/Himes/Documents/.Dev/Python/ass_1_2.py
PS C:\Users\Himes\Documents\.Dev\Python>
```

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:

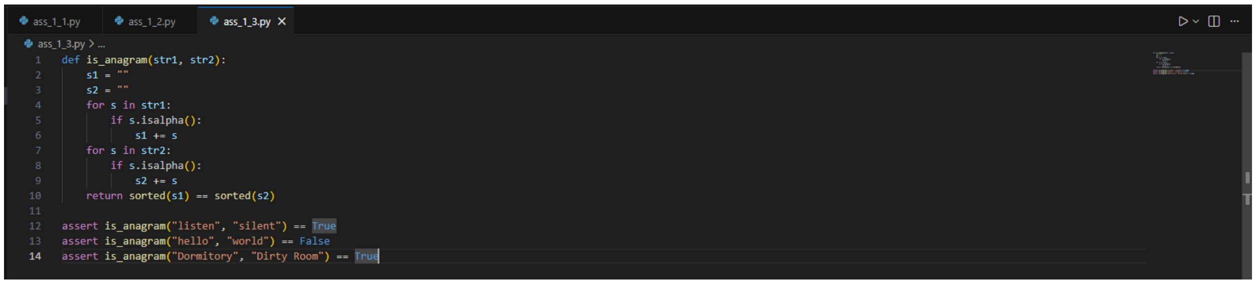
- Ignore case, spaces, and punctuation.
- Handle edge cases (empty strings, identical words).

Example Assert Test Cases:

- assert is_anagram("listen", "silent") == True
- assert is_anagram("hello", "world") == False
- assert is_anagram("Dormitory", "Dirty Room") == True

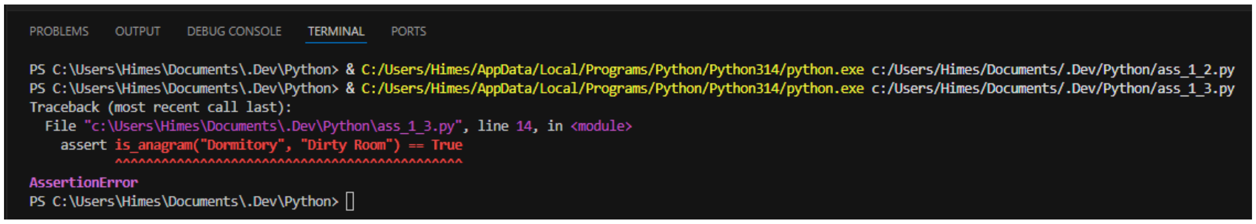
Expected Output:

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



```python
def is_anagram(str1, str2):
    s1 = ""
    s2 = ""
    for s in str1:
        if s.isalpha():
            s1 += s
    for s in str2:
        if s.isalpha():
            s2 += s
    return sorted(s1) == sorted(s2)

assert is_anagram("listen", "silent") == True
assert is_anagram("hello", "world") == False
assert is_anagram("Dormitory", "Dirty Room") == True
```

Output:



```
PS C:\Users\Himes\Documents\.Dev\Python> & C:/Users/Himes/AppData/Local/Programs/Python/Python314/python.exe c:/Users/Himes/Documents/.Dev/Python/ass_1_2.py
PS C:\Users\Himes\Documents\.Dev\Python> & C:/Users/Himes/AppData/Local/Programs/Python/Python314/python.exe c:/Users/Himes/Documents/.Dev/Python/ass_1_3.py
Traceback (most recent call last):
  File "c:\Users\Himes\Documents\.Dev\Python\ass_1_3.py", line 14, in <module>
    assert is_anagram("Dormitory", "Dirty Room") == True
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

AssertionError
PS C:\Users\Himes\Documents\.Dev\Python>
```

4. (Inventory Class – Apply AI to Simulate Real-World Inventory System)

Task:

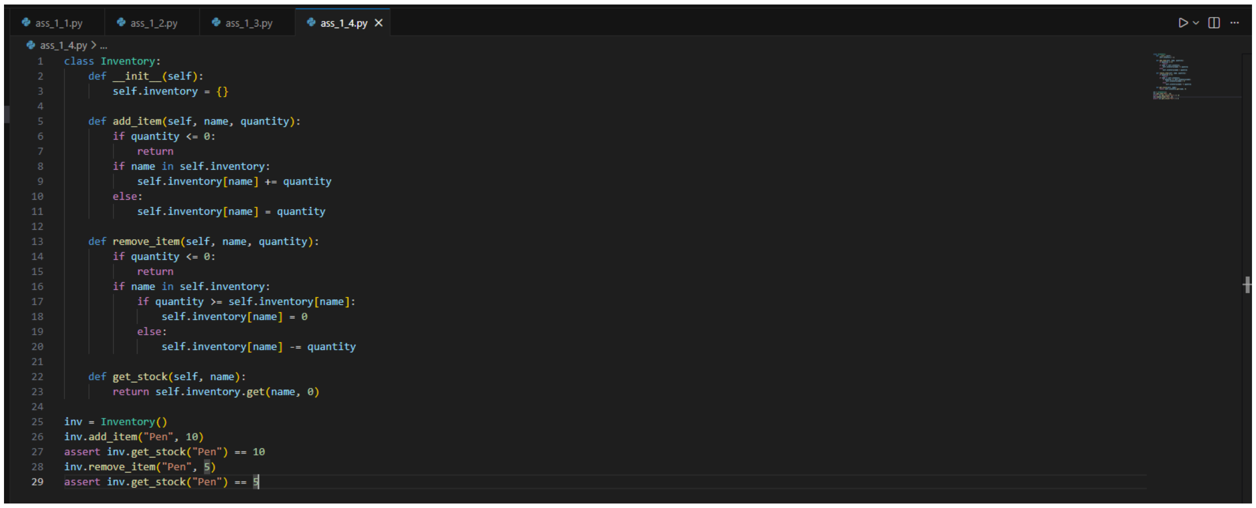
Ask AI to generate at least 3 assert-based tests for an Inventory class with stock management.

Methods:

- add_item(name, quantity)
- remove_item(name, quantity)
- get_stock(name)

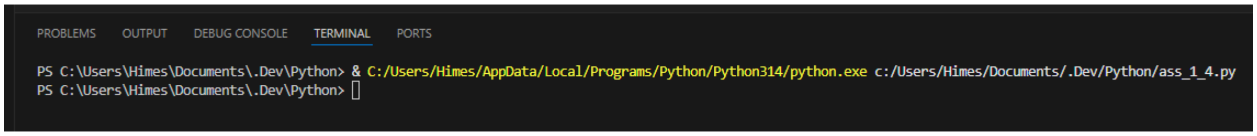
Example Assert Test Cases:

- inv = Inventory()
- inv.add_item("Pen", 10)
- assert inv.get_stock("Pen") == 10
- inv.remove_item("Pen", 5)
- assert inv.get_stock("Pen") == 5



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

    def add_item(self, name, quantity):
        if quantity <= 0:
            return
        if name in self.inventory:
            self.inventory[name] += quantity
        else:
            self.inventory[name] = quantity

    def remove_item(self, name, quantity):
        if quantity <= 0:
            return
        if name in self.inventory:
            if quantity >= self.inventory[name]:
                self.inventory[name] = 0
            else:
                self.inventory[name] -= quantity

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

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

Output:



```
PS C:\Users\Himes\Documents\.Dev\Python> & C:/Users/Himes/AppData/Local/Programs/Python/Python314/python.exe c:/Users/Himes/Documents/.Dev/Python/ass_1_4.py
PS C:\Users\Himes\Documents\.Dev\Python> 
```

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:

- Validate "MM/DD/YYYY" format.
- Handle invalid dates.
- Convert valid dates to "YYYY-MM-DD".

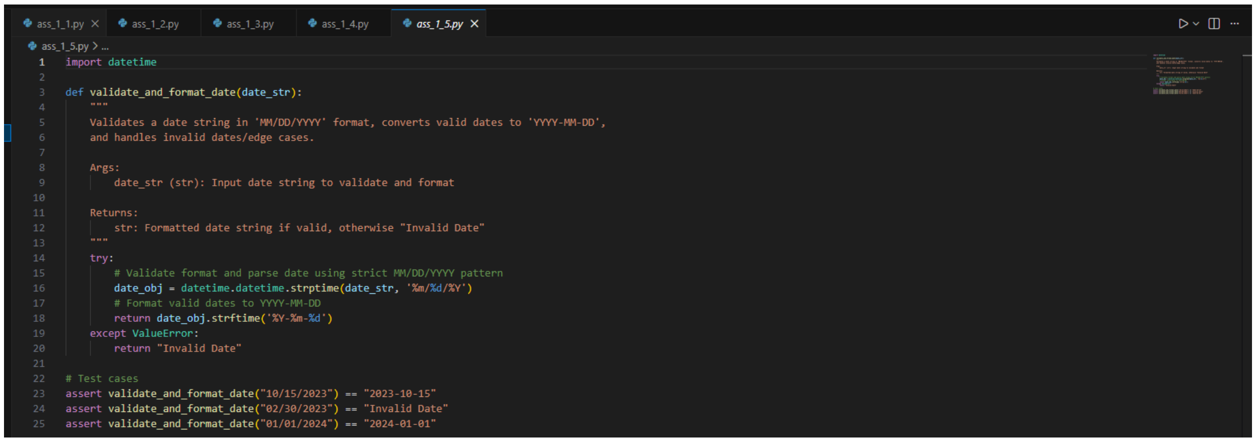
Example Assert Test Cases:

- 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"

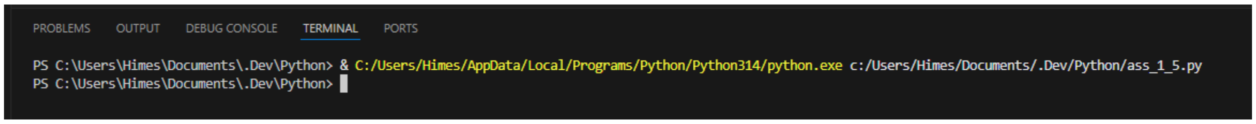
Expected Output:

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



```python
import datetime

def validate_and_format_date(date_str):
    """
    Validates a date string in 'MM/DD/YYYY' format, converts valid dates to 'YYYY-MM-DD',
    and handles invalid dates/edge cases.

    Args:
        date_str (str): Input date string to validate and format

    Returns:
        str: Formatted date string if valid, otherwise "Invalid Date"
    """
    try:
        # Validate format and parse date using strict MM/DD/YYYY pattern
        date_obj = datetime.datetime.strptime(date_str, '%m/%d/%Y')
        # Format valid dates to YYYY-MM-DD
        return date_obj.strftime('%Y-%m-%d')
    except ValueError:
        return "Invalid Date"

# 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:



```
PS C:\Users\Himes\Documents\.Dev\Python> & C:/Users/Himes/AppData/Local/Programs/Python/Python314/python.exe c:/Users/Himes/Documents/.Dev/Python/ass_1_5.py
PS C:\Users\Himes\Documents\.Dev\Python> 
```