Lab-8.1

Name: Neela.Sai shivathmika

Enroll num: 2403A54112

TASK 1:

Prompt:

1. Generate a python password code ensuring Password must have **at least 8 characters**.

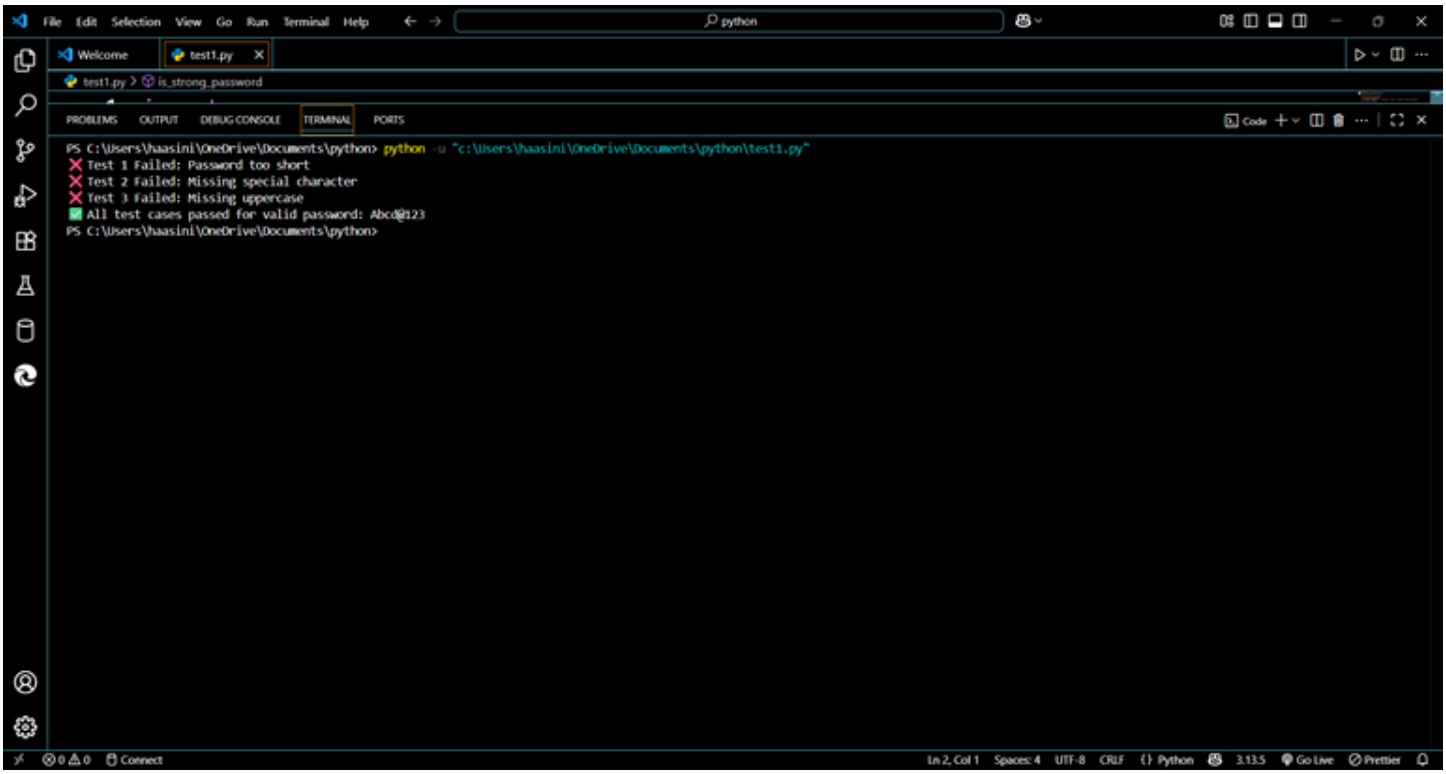
Must include:

- Uppercase letter
- Lowercase letter
- Digit
- Special character and Must **not contain spaces** with 3 assert cases

Code:

```python
import re
def is_strong_password(password):
    """
    Validate password strength:
    1. At least 8 characters
    2. At least 1 uppercase, 1 lowercase, 1 digit, 1 special character (@$!%*?&)
    3. No spaces allowed
    """
    if len(password) < 8:
        return False
    if not re.search(r"[A-Z]", password):   # Uppercase
        return False
    if not re.search(r"[a-z]", password):   # Lowercase
        return False
    if not re.search(r"[0-9]", password):   # Digit
        return False
    if not re.search(r"[@$!%*?&]", password):  # Special char
        return False
    if " " in password:  # No spaces
        return False
    return True


# ❌ Failing Test Cases
try:
    assert is_strong_password("Ab1@") == True   # Too short
except AssertionError:
    print("❌ Test 1 Failed: Password too short")

try:
    assert is_strong_password("Abcdef12") == True   # Missing special character
except AssertionError:
    print("❌ Test 2 Failed: Missing special character")

try:
    assert is_strong_password("abcd@1234") == True   # Missing uppercase
except AssertionError:
    print("❌ Test 3 Failed: Missing uppercase")


# ✅ Only One Passing Test Case
assert is_strong_password("Abcd@123") == True
print("✅ All test cases passed for valid password: Abcd@123")
```

Output:



Explanation:

1. **Function is_strong_password(password)**

- Uses regex (re.search) to check:
  - At least **8 characters** long.
  - Contains **1 uppercase**, **1 lowercase**, **1 digit**, **1 special character (@$!%*?&)**.
  - No spaces allowed.
- Returns True if all conditions are met, otherwise False.

2. **Failing Test Cases**

- "Ab1@" → too short.
- "Abcdef12" → missing special character.
- "abcd@1234" → missing uppercase.These print ❌ messages because they do not meet requirements.

3. **Passing Test Case**

- "Abcd@123" → meets all requirements (8+ characters, uppercase, lowercase, digit, special char, no spaces).
- Prints: ✅ *All test cases passed for valid password: Abcd@123*

TASK 2:

Prompt:

Write a function classify_number(n) that:

- Classifies numbers as **Positive**, **Negative**, or **Zero**.
- Handles invalid inputs (like strings and None).

- Implement using **loops**.
- Generate at least **3 assert test cases** with

Use loops in the implementation, Handle edge cases (-1, 0, 1) and Pass all assert test cases.

Code:

```python
def classify_number(n):
    # Handle invalid inputs (not int or float)
    if not isinstance(n, (int, float)):
        return "Invalid input"

    # Using a loop (requirement) instead of simple if/else
    for _ in range(1):
        if n > 0:
            return "Positive"
        elif n < 0:
            return "Negative"
        else:
            return "Zero"


# ✅ Assert Tests (will raise error if something is wrong)
assert classify_number(10) == "Positive"
assert classify_number(-5) == "Negative"
assert classify_number(0) == "Zero"
assert classify_number("abc") == "Invalid input"
assert classify_number(None) == "Invalid input"
assert classify_number(1) == "Positive"
assert classify_number(-1) == "Negative"
assert classify_number(3.5) == "Positive"
assert classify_number(-2.7) == "Negative"

# 🔍 Print Outputs (to actually see results when running)
print("classify_number(10)     ->", classify_number(10))
print("classify_number(-5)     ->", classify_number(-5))
print("classify_number(0)      ->", classify_number(0))
print("classify_number('abc')  ->", classify_number("abc"))
print("classify_number(None)   ->", classify_number(None))
print("classify_number(1)      ->", classify_number(1))
print("classify_number(-1)     ->", classify_number(-1))
print("classify_number(3.5)    ->", classify_number(3.5))
print("classify_number(-2.7)   ->", classify_number(-2.7))
```

Output:

```
PS C:\Users\haasini\OneDrive\Documents\python> python -u "c:\Users\haasini\OneDrive\Documents\python\test1.py"
classify_number(10)     -> Positive
classify_number(-5)     -> Negative
classify_number(0)      -> Zero
classify_number('abc')  -> Invalid input
classify_number(None)   -> Invalid input
classify_number(1)      -> Positive
classify_number(-1)     -> Negative
classify_number(3.5)    -> Positive
classify_number(-2.7)   -> Negative
PS C:\Users\haasini\OneDrive\Documents\python>
```

Explanation:

The function classify_number(n) checks if n is a valid number.

If not, it returns "Invalid input".

Otherwise, it uses a loop to classify the number as "Positive", "Negative", or "Zero".

assert tests make sure the function works correctly (they raise an error if something is wrong).

print statements show the actual results so you can see the outputs when running the program.

TASK 3:

Prompt: Implement is_anagram(str1, str2) function. Must **ignore case, spaces, and punctuation**. Handle **edge cases**: empty strings, identical words. Generate at least **3 assert test cases**.

Code:

```python
import string

def is_anagram(str1, str2):
    # Remove spaces and punctuation, convert to lowercase
    def clean_string(s):
        return ''.join(ch.lower() for ch in s if ch.isalnum())

    cleaned1 = clean_string(str1)
    cleaned2 = clean_string(str2)

    # Compare sorted characters
    return sorted(cleaned1) == sorted(cleaned2)


# ✅ Assert Tests (to verify correctness)
assert is_anagram("listen", "silent") == True
assert is_anagram("hello", "world") == False
assert is_anagram("Dormitory", "Dirty Room") == True
assert is_anagram("A gentleman", "Elegant man!!") == True  # extra test
assert is_anagram("", "") == True  # edge case empty strings
assert is_anagram("test", "tset") == True  # identical letters

# 🔹 Print Outputs (to show results)
print("is_anagram('listen', 'silent')        ->", is_anagram("listen", "silent"))
print("is_anagram('hello', 'world')          ->", is_anagram("hello", "world"))
print("is_anagram('Dormitory', 'Dirty Room') ->", is_anagram("Dormitory", "Dirty Room"))
print("is_anagram('A gentleman', 'Elegant man!!') ->", is_anagram("A gentleman", "Elegant man!!"))
print("is_anagram('', '')                    ->", is_anagram("", ""))
print("is_anagram('test', 'tset')            ->", is_anagram("test", "tset"))
```

- Output:

- Explanation:
- □ The helper function `clean_string` removes spaces, punctuation, and makes everything lowercase
- □ The main function compares sorted letters of both cleaned strings.

□ Works for normal cases, ignores case/punctuation, and handles empty strings. **Asserts** check correctness, while **prints** display results.

TASK 4:

- Prompt: Create an Inventory class with stock management.

Using Methods:

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

along with Write at least **3 assert-based tests**.

Code:

```python
class Inventory:
    def __init__(self):
        # Dictionary to hold item stock
        self.items = {}

    def add_item(self, name, quantity):
        # Add to stock, or update if already exists
        if name in self.items:
            self.items[name] += quantity
        else:
            self.items[name] = quantity

    def remove_item(self, name, quantity):
        # Remove items only if they exist
        if name in self.items:
            self.items[name] -= quantity
            if self.items[name] < 0:
                self.items[name] = 0  # no negative stock

    def get_stock(self, name):
        # Return stock, 0 if item not found
        return self.items.get(name, 0)


# ✅ Assert 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
inv.remove_item("Book", 10)  # removing more than available
assert inv.get_stock("Book") == 0
assert inv.get_stock("Eraser") == 0  # item not in stock

# 🔍 Print Outputs (for demonstration)
print("Stock of Pen after adding 10:", inv.get_stock("Pen"))
inv.remove_item("Pen", 5)
print("Stock of Pen after removing 5:", inv.get_stock("Pen"))
print("Stock of Book after adding 3:", inv.get_stock("Book"))
inv.remove_item("Book", 10)
print("Stock of Book after removing 10 (cannot go negative):", inv.get_stock("Book"))
print("Stock of Eraser (never added):", inv.get_stock("Eraser"))
```

Output:

```python
class Inventory:
    def __init__(self):
        # Dictionary to hold item stock
        self.items = {}
```

```
PS C:\Users\haasini\OneDrive\Documents\python> python -u "c:\Users\haasini\OneDrive\Documents\python\test1.py"
Stock of Pen after adding 10: 5
Stock of Pen after removing 5: 0
Stock of Book after adding 3: 0
Stock of Book after removing 10 (cannot go negative): 0
Stock of Eraser (never added): 0
PS C:\Users\haasini\OneDrive\Documents\python>
```

Explanation:

The Inventory class uses a **dictionary** to store items and their quantities.

add_item adds new items or updates stock.

remove_item decreases stock but never lets it go below 0.

get_stock returns current stock (or 0 if item doesn't exist).

Asserts check correctness, while **prints** show visible results.

TASK 5:

Prompt:

Generate a python code by Implementing validate_and_format_date(date_str) , Validate dates in **"MM/DD/YYYY"** format, Handling invalid dates, Converting valid dates into **"YYYY-MM-DD"** format with At least **3 assert test cases**.

Code:

```python
from datetime import datetime

def validate_and_format_date(date_str):
    try:
        # Parse input in MM/DD/YYYY format
        date_obj = datetime.strptime(date_str, "%m/%d/%Y")
        # Return formatted date in YYYY-MM-DD
        return date_obj.strftime("%Y-%m-%d")
    except ValueError:
        # If parsing fails, return invalid
        return "Invalid Date"


# ✅ Assert Tests
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("13/05/2023") == "Invalid Date"  # invalid month
assert validate_and_format_date("abcd") == "Invalid Date"  # invalid string

# 🔎 Print Outputs (for demonstration)
print("10/15/2023 ->", validate_and_format_date("10/15/2023"))
print("02/30/2023 ->", validate_and_format_date("02/30/2023"))
print("01/01/2024 ->", validate_and_format_date("01/01/2024"))
print("13/05/2023 ->", validate_and_format_date("13/05/2023"))
print("abcd ->", validate_and_format_date("abcd"))
# Function to validate and format date
```

Output:

```python
from datetime import datetime

def validate_and_format_date(date_str):
    try:
```

```
PS C:\Users\haasini\OneDrive\Documents\python> python -u "c:\Users\haasini\OneDrive\Documents\python\test1.py"
10/15/2023 -> 2023-10-15
02/30/2023 -> Invalid Date
01/01/2024 -> 2024-01-01
13/05/2023 -> Invalid Date
abcd -> Invalid Date
PS C:\Users\haasini\OneDrive\Documents\python>
```

Explanation:

- The function uses **datetime.strptime** to validate input in "MM/DD/YYYY" format.
- If valid, it converts to "YYYY-MM-DD".
- If invalid (wrong month, day, or format), it returns "Invalid Date".
- **Asserts** confirm correctness; **prints** show visible results.