LAB ASSIGNMENT – 8.1

NAME: CH VISHWANATH RAO

BATCH: 02

HT NO: 2303A51095

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.

CODE:

```python
def checkthepassword(password):
    if len(password) < 8:
        return False
    if not any(char.isupper() for char in password):
        return False
    if not any(char.islower() for char in password):
        return False
    if not any(char.isdigit() for char in password):
        return False
    if not any(char in "!@#$%^&*()-_=+[]{}|;:'\",.<>?/" for char in password):
        return False
    return True
```

```python
def checkthepassword(password):
    if len(password) < 8:
        return False
    if not any(char.isupper() for char in password):
        return False
    if not any(char.islower() for char in password):
        return False
    if not any(char.isdigit() for char in password):
        return False
    if not any(char in "!@#$%^&*()-_=+[]{}|;:'\",.<>?/" for char in password):
        return False
    return True
#assert testcases
assert checkthepassword("Password123!") == True
assert checkthepassword("pass") == False
assert checkthepassword("PASSWORD123") == False
assert checkthepassword("password123") == False
assert checkthepassword("Password") == False
assert checkthepassword("Password123") == False
assert checkthepassword("Password!") == False
assert checkthepassword("12345678") == False
assert checkthepassword("!@#$%^&*") == False
print("All test cases passed!")
```

OUTPUT:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\chvis\OneDrive\Desktop\ai> & C:/Python314/python.exe c:/Users/chvis/OneDrive/Desktop/ai/DOC1.py
All test cases passed!
PS C:\Users\chvis\OneDrive\Desktop\ai>
```

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.

CODE:

```python
def define_number(n):
    if n>0:
        return "+ve"
    elif n<0:
        return "-ve"
    else:
        return "0"
assert define_number(10) == "+ve"
assert define_number(-5) == "-ve"
assert define_number(0) == "0"
print("All test cases passed!")
```

OUTPUT:

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.

CODE:

```
def is_anagram(str1, str2):
    return sorted(str1.replace(" ", "").lower()) == sorted(str2.replace(" ", "").lower())
assert is_anagram("listen", "silent") == True
assert is_anagram("hello", "world") == False
assert is_anagram("Dormitory", "Dirty Room") == True
assert is_anagram("The eyes", "They see") == True
assert is_anagram("Astronomer", "Moon starer") == True
assert is_anagram("Conversation", "Voices rant on") == True
print("All test cases passed!")
def is_palindrome(word):
    return word.lower() == word[::-1].lower()
assert is_palindrome("racecar") == True
assert is_palindrome("hello") == False
assert is_palindrome("Was it a car or a cat I saw?") == True
assert is_palindrome("A man, a plan, a canal, Panama!") == True
assert is_palindrome("No 'x' in Nixon") == True
print("All test cases passed!")
```

Output:

```
All test cases passed!
PS C:\Users\chvis\OneDrive\Desktop\ai>
```

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.

CODE:

```python
class Inventory:
    def __init__(self):
        self.p = {}
    def add(self, name, n):
        self.p[name] = self.p.get(name, 0) + n
    def remove(self, name, n):
        if self.p.get(name, 0) < n: raise ValueError("Not enough quantity")
        self.p[name] -= n
        if not self.p[name]: del self.p[name]
    def get(self, name):
        return self.p.get(name, 0)

w = Inventory()
for item, qty in [("apple", 10), ("banana", 5)]:
    w.add(item, qty)
    assert w.get(item) == qty
w.remove("apple", 3)
assert w.get("apple") == 7
w.remove("banana", 5)
assert w.get("banana") == 0
print("All test cases passed!")
```

Output:

```
PS C:\Users\chvis\OneDrive\Desktop\ai> & C:/Python314/python.exe c:/Users/chvis/OneDrive/Desktop/ai/DOC1.py
All test cases passed!
```

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.

CODE:

```python
import re
from datetime import datetime

def chk_date(d):
    if not re.match(r'^\d{2}/\d{2}/\d{4}$', d):
        return "Invalid Date"
    try:
        return datetime.strptime(d, "%m/%d/%Y").strftime("%Y-%m-%d")
    except ValueError:
        return "Invalid Date"

assert chk_date("12/31/2020") == "2020-12-31"
assert chk_date("31/12/2020") == "Invalid Date"
assert chk_date("02/30/2020") == "Invalid Date"
assert chk_date("01/01/2021") == "2021-01-01"
assert chk_date("13/01/2020") == "Invalid Date"
print("All test cases passed!")
```

OUTPUT:

```
PS C:\Users\chvis\OneDrive\Desktop\ai> & C:/Python314/python.exe c:/Users/chvis/OneDrive/Desktop/ai/DOC1.py
All test cases passed!
```