ASSIGNMENT-8.1

T.Rakshitha

2303A51172

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

PROMPT : #generate a code for password strength validator password must be at least 8 characters long, contain at least one uppercase letter, one lowercase letter, one digit, and one special character

CODE :

```
import re
def validate_password(password):
    if len(password) < 8:
        return "Password must be at least 8 characters long."
    if not re.search(r'[A-Z]', password):
        return "Password must contain at least one uppercase letter."
    if not re.search(r'[a-z]', password):
        return "Password must contain at least one lowercase letter."
    if not re.search(r'[0-9]', password):
        return "Password must contain at least one digit."
    if not re.search(r'[!@#$%^&*(),.?":{}|<>]', password):
        return "Password must contain at least one special character."
    return "Password is strong."
# Example usage
password = input("Enter a password to validate: ")
```

```
result = validate_password(password)
print(result)
```

PROMPT 2 : #if it doesnot meet ask again to enter the password

CODE :

```
while True:
    password = input("Enter a password to validate: ")
    result = validate_password(password)
    print(result)
    if result == "Password is strong.":
        break
```

OUTPUT ;

```
Enter a password to validate: vishnu@06
Password must contain at least one uppercase letter.
Enter a password to validate: Vishnu@06
Password is strong.
```

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.

Prompt : #generate a code for classification of numbers as positive, negative, or zero handle invalid inputs like strings or special characters

Code :

```
def classify_number(num):
    try:
        number = float(num)
        if number > 0:
```

```
        return "The number is positive."
    elif number < 0:
        return "The number is negative."
    else:
        return "The number is zero."
  except ValueError:
    return "Invalid input. Please enter a valid number."
# Example usage
while True:
  user_input = input("Enter a number to classify: ")
  result = classify_number(user_input)
  print(result)
  if "Invalid input" not in result:
    break
```

OUTPUT :

```
Enter a number to classify: -6
The number is negative.
```

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

PROMPT : #GENERATE A CODE FOR ANAGTRAM checker that takes two strings as input and checks if they are anagrams of each other, ignoring spaces and case sensitivity

CODE :

```
def are_anagrams(str1, str2):


    str1 = str1.replace(" ", "").lower()
    str2 = str2.replace(" ", "").lower()


    # Sort the characters of both strings and compare
    return sorted(str1) == sorted(str2)
# Example usage
string1 = input("Enter the first string: ")
string2 = input("Enter the second string: ")
if are_anagrams(string1, string2):
    print("The strings are anagrams of each other.")
else:
    print("The strings are not anagrams of each other.")
```

PROMPT -2 : # ask the user to enter two strings until they are anagrams of each other

CODE :

```
while True:
    string1 = input("Enter the first string: ")
    string2 = input("Enter the second string: ")
    if are_anagrams(string1, string2):
        print("The strings are anagrams of each other.")
        break
    else:
        print("The strings are not anagrams of each other. Please try again.")
```

OUTPUT :

```
Enter the first string: VISHNU
Enter the second string: MY SELF
The strings are not anagrams of each other.
Enter the first string: LISTEN
Enter the second string: SILENT
The strings are anagrams of each other.
```

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.

PROMPT : #Inventory Class – Apply AI to Simulate Real-World Inventory System AND generate at least 3 assert-based tests for an Inventory class with stock management TAKE USER INPUT TO TEST THE CLASS METHODS AND PRINT THE RESULTS OF THE TESTS. The Inventory class should have methods to add items, remove items, and check stock levels. The tests should cover scenarios such as adding items, removing items, and checking stock levels for existing and non-existing items.

Code :

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

    def add_item(self, item, quantity):
        if item in self.stock:
            self.stock[item] += quantity
        else:
            self.stock[item] = quantity
```

```
    def remove_item(self, item, quantity):
        if item in self.stock and self.stock[item] >= quantity:
            self.stock[item] -= quantity
            return True
        return False

    def check_stock(self, item):
        return self.stock.get(item, 0)
# take user inputs to test the Inventory class methods
inventory = Inventory()
# Test adding items
item_to_add = input("Enter the item to add: ")
quantity_to_add = int(input("Enter the quantity to add: "))
inventory.add_item(item_to_add, quantity_to_add)
# Test checking stock levels
item_to_check = input("Enter the item to check stock levels: ")
stock_level = inventory.check_stock(item_to_check)
print(f"Stock level for {item_to_check}: {stock_level}")
# Test removing items
item_to_remove = input("Enter the item to remove: ")
quantity_to_remove = int(input("Enter the quantity to remove: "))
if inventory.remove_item(item_to_remove, quantity_to_remove):
    print(f"Removed {quantity_to_remove} of {item_to_remove}.")
else:
    print(f"Failed to remove {quantity_to_remove} of {item_to_remove}. Not enough stock or item does not exist.")
# Assert-based tests
```

assert inventory.check_stock(item_to_add) == quantity_to_add, "Test failed: Item not added correctly."

assert inventory.remove_item(item_to_add, quantity_to_add) == True, "Test failed: Item not removed correctly."

assert inventory.check_stock(item_to_add) == 0, "Test failed: Stock level not updated after removal."

assert inventory.remove_item(item_to_add, 1) == False, "Test failed: Should not remove item that is out of stock."

print("All tests passed successfully.")

OUTPUT :

```
Enter the item to add: apple
Enter the quantity to add: 12
Enter the item to check stock levels: apple
Stock level for apple: 12
Enter the item to remove: mango
Enter the quantity to remove: 12
Failed to remove 12 of mango. Not enough stock or item does not exist.
All tests passed successfully.
```

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.

Prompt : #DATE VAlidation code that takes a date input from the user in the format "DD/MM/YYYY" and checks if it is a valid date, considering leap years and the number of days in each month. If the date is valid, print "Valid date." Otherwise, print "Invalid date." The code should also handle invalid inputs such as non-numeric characters or incorrect formats.

Code :

import re

def validate_date(date_str):

```
    # Check if the format is correct
    if not re.match(r'^\d{2}/\d{2}/\d{4}$', date_str):
        return "Invalid date format. Please use DD/MM/YYYY."

    day, month, year = map(int, date_str.split('/'))

    # Check for valid month
    if month < 1 or month > 12:
        return "Invalid month. Month must be between 1 and 12."

    # Check for valid day based on the month
    if month in [1, 3, 5, 7, 8, 10, 12]:  # Months with 31 days
        if day < 1 or day > 31:
            return "Invalid day. Day must be between 1 and 31 for the given month."
    elif month in [4, 6, 9, 11]:  # Months with 30 days
        if day < 1 or day > 30:
            return "Invalid day. Day must be between 1 and 30 for the given month."
    else:  # February
        if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):  # Leap year
            if day < 1 or day > 29:
                return "Invalid day. Day must be between 1 and 29 for February in a leap year."
        else:
            if day < 1 or day > 28:
                return "Invalid day. Day must be between 1 and 28 for February in a non-leap year."

    return "Valid date."
# Example usage
```

```
date_input = input("Enter a date (DD/MM/YYYY): ")
result = validate_date(date_input)
print(result)
```

#ASK UNTIL THE USER ENTERS A VALID DATE

```
while True:
    date_input = input("Enter a date (DD/MM/YYYY): ")
    result = validate_date(date_input)
    print(result)
    if result == "Valid date.":
        break
```

OUTPUT :