Lab Assignment-5.1

G Datha narayana

2303A51754

Batch-11

Task Description #1 (Privacy in API Usage):

Task: Use an AI tool to generate a Python program that connects to a weather API.

Prompt:

#Generate code to fetch weather data securely without exposing API keys in the code.

Code:

```
lab1 > lab1 > lab 5.1.py > get_weather
#Generate code to fetch weather data securely without exposing API keys in the code.
import os
import requests
from dotenv import load_dotenv
load_dotenv()
API_KEY = os.getenv('WEATHER_API_KEY')
BASE_URL = "http://api.openweathermap.org/data/2.5/weather"
def get_weather(city):
    params = {
        'q': city,
        'appid': API_KEY,
        'units': 'metric'
    }
    response = requests.get(BASE_URL, params=params)
    if response.status_code == 200:
        return response.json()
    else:
        return None
if __name__ == "__main__":
    city = input("Enter city name: ")
    weather_data = get_weather(city)
    if weather_data:
        print(f"Weather in {city}: {weather_data['weather'][0]['description']}, Temperature: {weather_data['main']['temp']}°C")
    else:
        print("City not found or error fetching data.")
```

Output:

```
Enter city name: london
Weather in london: broken clouds, Temperature: 9.56°C
```

Explanation:

This Python script demonstrates how to securely fetch weather data for a user-specified city using the OpenWeatherMap API. It begins by loading environment variables from a .env file, ensuring the API key is not exposed directly in the code. The get_weather function constructs a request with the city name and API key, sending it to the weather API and returning the result as JSON if successful. The main block prompts the user to enter a city name, retrieves the weather data, and displays the weather description and temperature if found. If the city is not found or an error occurs, an appropriate error message is shown, making the script both secure and user-friendly.

Task Description #2 (Privacy & Security in File Handling)

Task: Use an AI tool to generate a Python script that stores user data (name, email, password) in a file.

Prompt:

```python
#create a python script that stores user data (name,email,password).
import json
def store_user_data(name, email, password):
    user_data = {
        'name': name,
        'email': email,
        'password': password
    }
    with open('user_data.json', 'w') as f:
        json.dump(user_data, f)
if __name__ == "__main__":
    name = input("Enter your name: ")
    email = input("Enter your email: ")
    password = input("Enter your password: ")
    store_user_data(name, email, password)
    print("User data stored successfully.")
```

Output:

```
{} user_data.json > ...
1   {"name": "mosin", "email": "mosin@gmail.com", "password": "mosin909090"}
```

With Encrypted:

```python
#create a python script that stores user data (name,email,password) with encrypted password.
import json
import hashlib
def store_user_data(name, email, password):
    hashed_password = hashlib.sha256(password
.encode()).hexdigest()
    user_data = {
        'name': name,
        'email': email,
        'password': hashed_password
    }
    with open('user_data_encrypted.json', 'w') as f:
        json.dump(user_data, f)
if __name__ == "__main__":
    name = input("Enter your name: ")
    email = input("Enter your email: ")
    password = input("Enter your password: ")
    store_user_data(name, email, password)
    print("User data with encrypted password stored successfully.")
```

Output:

```
{} user_data_encrypted.json > ...
1    {"name": "mesin", "email": "mesin@gmail.com", "password": "17450e00f404d31cb8546a7dc95d669d2c220d6d975fff691ea406e0a1196169"}
```

Explanation:

These two Python scripts demonstrate how to store user data securely. The first script collects a user's name, email, and password, then saves this information in a JSON file called user_data.json. The second script improves security by encrypting the password using the SHA-256 hashing algorithm before saving it, so the actual password is not stored in plain text. This hashed password, along with the user's name and email, is saved in user_data_encrypted.json. Both scripts prompt the user for their details and confirm successful storage, but the second script ensures that sensitive password information is protected from unauthorized access.

Task Description #3 (Transparency in Algorithm Design)

Objective: Use AI to generate an Armstrong number checking function with comments and explanations.

Prompt:

#armstrong number checking program

Code:

```python
#armstrong number checking program
def is_armstrong_number(num):
    # Convert number to string to easily iterate over digits
    num_str = str(num)
    # Calculate the number of digits
    num_digits = len(num_str)
    # Calculate the sum of each digit raised to the power of the number of digits
    total = sum(int(digit) ** num_digits for digit in num_str)
    # Check if the total equals the original number
    return total == num
# Test the function
if __name__ == "__main__":
    number = int(input("Enter a number: "))
    if is_armstrong_number(number):
        print(f"{number} is an Armstrong number.")
    else:
        print(f"{number} is not an Armstrong number.")
```

Output:

```
PS C:\Users\Awez_\OneDrive\Desktop\Ai coding>
Enter a number: 152
152 is not an Armstrong number.
```

```
#explanation of code line by line
The function is_armstrong_number takes an integer num as input.
It converts the number to a string to easily iterate over each digit.
It calculates the number of digits in the number.
It computes the sum of each digit raised to the power of the number of digits using a
generator expression inside the sum function.
Finally, it checks if the computed total is equal to the original number and returns True or False.
```

Explanation:

This code defines a function to check if a number is an Armstrong number, which means the sum of its digits each raised to the power of the number of digits equals the number itself. The function converts the number to a string to easily iterate over its digits, calculates the number of digits, and computes the required sum. In the main block, it prompts the user to enter a number, checks if it is an Armstrong number using the function, and prints the result. This approach works for any integer, not just three-digit numbers.

Task Description #4 (Transparency in Algorithm Comparison)

Task: Use AI to implement two sorting algorithms (e.g., QuickSort and BubbleSort).

Prompt:

#Generate Python code for QuickSort and BubbleSort, and include comments explaining step-by-step how each works and where they differ.

Code:

```python
#Generate Python code for QuickSort and BubbleSort, and include comments explaining step-by-step how each works and where they differ.
# QuickSort Algorithm
def quicksort(arr):
    # Base case: if the array is empty or has one element, it's already sorted
    if len(arr) <= 1:
        return arr
    else:
        # Choose a pivot element (here we choose the last element)
        pivot = arr[-1]
        # Create two lists to hold elements less than and greater than the pivot
        less_than_pivot = []
        greater_than_pivot = []
        # Iterate through all elements except the pivot
        for element in arr[:-1]:
            if element <= pivot:
                less_than_pivot.append(element)  # Elements less than or equal to pivot
            else:
                greater_than_pivot.append(element)  # Elements greater than pivot
        # Recursively apply quicksort to the less and greater lists, and combine with pivot
        return quicksort(less_than_pivot) + [pivot] + quicksort(greater_than_pivot)
# BubbleSort Algorithm
def bubblesort(arr):
    n = len(arr)
    # Traverse through all array elements
    for i in range(n):
        # Last i elements are already sorted, no need to check them
        for j in range(0, n-i-1):
            # Swap if the element found is greater than the next element
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr
# Example usage
if __name__ == "__main__":
    sample_array = [64, 34, 25, 12, 22, 11, 90]
    print("Original array:", sample_array)
    print("Sorted array using QuickSort:", quicksort(sample_array))
    print("Sorted array using BubbleSort:", bubblesort(sample_array.copy()))
```

```
#Transparent, comparative explanation of their logic and efficiency in points.
# QuickSort:
1. QuickSort is a divide-and-conquer algorithm that selects a 'pivot'
2. It partitions the array into two sub-arrays: elements less than the pivot and elements greater than the pivot.
3. It recursively sorts the sub-arrays and combines them with the pivot to form a sorted array.
4. QuickSort has an average time complexity of O(n log n) but can degrade to O(n^2) in the worst case (e.g., when the smallest or largest element is always chosen as the pivot).
# BubbleSort:
1. BubbleSort is a simple comparison-based algorithm that repeatedly steps through the list.
2. It compares adjacent elements and swaps them if they are in the wrong order.
3. This process is repeated until the list is sorted.
4. BubbleSort has a time complexity of O(n^2) in the average and worst cases, making it inefficient for large datasets compared to QuickSort.
```

Output:

```
PS C:\Users\Awez_\OneDrive\Desktop\Ai coding> python -u "c:\Users\Awe
Original array: [64, 34, 25, 12, 22, 11, 90]
Sorted array using QuickSort: [11, 12, 22, 25, 34, 64, 90]
Sorted array using BubbleSort: [11, 12, 22, 25, 34, 64, 90]
```

Explanation:

QuickSort is a divide-and-conquer algorithm that selects a pivot element and splits the array into smaller parts based on whether elements are smaller or larger than the pivot, then recursively sorts those parts. It is generally very fast for large datasets, with an average time complexity of $O(n \log n)$, but can degrade to $O(n^2)$ in the worst case if the pivot is poorly chosen. BubbleSort, on the other hand, works by repeatedly comparing adjacent elements and swapping them if they are in the wrong order until the entire list is sorted. It is simple and easy to understand but inefficient for large datasets due to its $O(n^2)$ time complexity. The key difference is that QuickSort is optimized for speed and scalability, while BubbleSort is better suited for learning and small input sizes because of its straightforward step-by-step approach.

Task Description #5 (Transparency in AI Recommendations)

Task: Use AI to create a product recommendation system.

Prompt:

#Generate a recommendation system that also provides reasons for each Suggestion

Code:

```python
#Generate a recommendation system that also provides reasons for each suggestion."
import random
class RecommendationSystem:
    def __init__(self, items):
        self.items = items

    def recommend(self, user_preferences, num_recommendations=5):
        recommendations = []
        for _ in range(num_recommendations):
            item = random.choice(self.items)
            reason = self.generate_reason(item, user_preferences)
            recommendations.append((item, reason))
        return recommendations

    def generate_reason(self, item, user_preferences):
        reasons = [
            f"This item matches your interest in {user_preferences}.",
            f"Based on your previous choices, you might like this item.",
            f"This item is popular among users with similar preferences.",
            f"You have shown interest in similar items before.",
            f"This item has high ratings in categories you prefer."
        ]
        return random.choice(reasons)
# Example usage
if __name__ == "__main__":
    items = [
        "Wireless Headphones",
        "Smart Watch",
        "E-book Reader",
        "Fitness Tracker",
        "Bluetooth Speaker",
        "Noise Cancelling Earbuds",
        "4K Action Camera",
        "Portable Charger",
        "Smart Home Hub",
        "VR Headset"
    ]
    user_preferences = "technology and gadgets"
    recommender = RecommendationSystem(items)
    recommendations = recommender.recommend(user_preferences)

    for item, reason in recommendations:
        print(f"Recommended Item: {item}\nReason: {reason}\n")
```

Output:

```
Reason: This item matches your interest in technology and gadgets.

Recommended Item: Smart Home Hub
Reason: This item matches your interest in technology and gadgets.

Recommended Item: Noise Cancelling Earbuds
Reason: Based on your previous choices, you might like this item.

Recommended Item: Noise Cancelling Earbuds
Reason: Based on your previous choices, you might like this item.
```

Explanation:

This recommendation system randomly selects products and generates a reason for each suggestion. It uses a class to store items and simulate personalized recommendations. The system matches user preferences with pre-defined reasoning messages to make suggestions feel intelligent. Although it does not use real AI or machine learning, it demonstrates the logic behind explainable recommendations. This model is useful for learning how real-world recommendation engines justify their results.