AI-Assisted Assingment 5.1 Using GitHub Copilot

Name: V.RaviTeja

Batch : 12

Rollno: 2303A51942

Task 1: Privacy in API Usage

Scenario

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

Copilot Prompt Used

```
"""Generate code to fetch weather data securely without exposing API keys
in the code """
```

Code :

```
import os
import requests
from dotenv import load_dotenv    "dotenv": Unknown word.
load_dotenv()    "dotenv": Unknown word.
API_KEY = os.getenv('WEATHER_API_KEY')
BASE_URL = "http://api.openweathermap.org/data/2.5/weather"
def get_weather(city):
    if not API_KEY:
        raise ValueError("API key not found. Please set the WEATHER_API_KEY environment variable.")
    params = {
        'q': city,
        'appid': API_KEY,
        'units': 'metric'
    }
    response = requests.get(BASE_URL, params=params)
    if response.status_code == 200:
        return response.json()
    else:
        response.raise_for_status()
if __name__ == "__main__":
    city = input("Enter city name: ")
    try:
        weather_data = get_weather(city)
        print(f"Weather in {city}: {weather_data['weather'][0]['description']}, Temperature: {weather_data['main']['temp']}°C")
    except Exception as e:
        print(f"Error fetching weather data: {e}")
```

Task 2: Privacy & Security in File Handling

Scenario

Use an AI tool to generate a Python script that stores user data (name, email, password) in a file Check if the AI stores sensitive data in plain text or without encryption. .

Copilot Prompt Used

```
"""Create a Python program to store user name, email, and password in a file.
Identify privacy risks in the program and rewrite it using secure password storage (hashing or encryption). """
```

Code :

```
import os
import hashlib
from getpass import getpass
from dotenv import load_dotenv    "dotenv": Unknown word.
load_dotenv()    "dotenv": Unknown word.
DATA_FILE = "user_data.txt"
def hash_password(password):
    """Hash a password for storing."""
    return hashlib.sha256(password.encode()).hexdigest()
def store_user_data(username, email, password):
    """Store user data securely in a file."""
    hashed_password = hash_password(password)
    with open(DATA_FILE, 'a') as f:
        f.write(f"{username},{email},{hashed_password}\n")

if __name__ == "__main__":
    username = input("Enter your username: ")
    email = input("Enter your email: ")
    password = getpass("Enter your password: ")
    store_user_data(username, email, password)
    print("User data stored securely.")
```

Sample Output

```
sathwik,sathwik@gmail.com,8bb0cf6eb9b17d0f7d22b456f121257dc1254e1f01665370476383ea776df414
```

```
PS D:\AI-Assisted coding> python -u "d:\AI-Assisted coding\Assingment5.1\Task2.py"
Enter your username: sathwik
Enter your email: sathwik@gmail.com
Enter your password:
User data stored securely.
PS D:\AI-Assisted coding>
```

Explanation

The slicing method simplifies the logic and removes unnecessary variables and loops.

The slicing method is faster in practice due to internal optimizations in Python.

Readability is significantly improved.

Task 3: Transparency in Algorithm Design)

Scenario

The string reversal logic is reused in multiple parts of an application

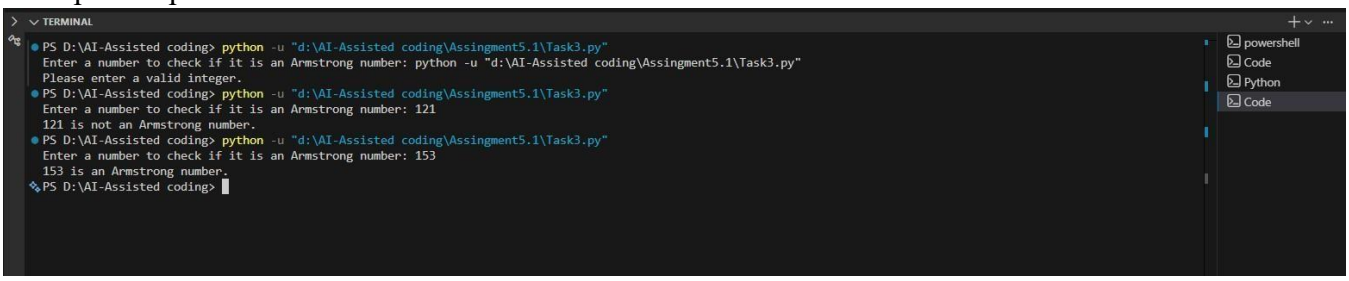
Copilot Prompt Used

```
# Generate a Python function that checks whether a given number is an Armstrong number.
# The code should include clear and meaningful comments explaining each step of the algorithm.
# Additionally, provide a simple, line-by-line explanation of how the code works to ensure transparency and easy understanding for beginners.
```

Code :

```
def is_armstrong_number(number):
    """
    Check if a number is an Armstrong number.

    An Armstrong number for a given number of digits is a number that is equal to the sum of its own digits each raised to the power of the number of digits.

    Parameters:
    number (int): The number to check.

    Returns:
    bool: True if the number is an Armstrong number, False otherwise.
    """
    # Convert the number to string to easily iterate over each digit
    digits = str(number)
    num_digits = len(digits)  # Get the number of digits in the number
    sum_of_powers = 0  # Initialize sum of powers

    # Iterate over each digit in the number
    for digit in digits:
        sum_of_powers += int(digit) ** num_digits  # Raise digit to the power of num_digits and add to sum

    # Check if the sum of powers is equal to the original number
    return sum_of_powers == number
# User input
try:
    user_input = int(input("Enter a number to check if it is an Armstrong number: "))
    if is_armstrong_number(user_input):
        print(f"{user_input} is an Armstrong number.")
    else:
        print(f"{user_input} is not an Armstrong number.")
except ValueError:
    print("Please enter a valid integer.")
```

Sample Output



Explanation

1. The function is_armstrong_number takes an integer input number.
2. It converts the number to a string to easily access each digit and calculates the number of digits. 3. It initializes a variable sum_of_powers to store the sum of each digit raised to the power of the number of digits.
4. It iterates through each digit, converts it back to an integer, raises it to the power of num_digits, and adds it to sum_of_powers.
5.Finally, it checks if sum_of_powers is equal to the original number and returns True or False accordingly.
6. The user is prompted to input a number, and the program checks if it is an Armstrong number, printing the result.

Task 4: Transparency in Algorithm Comparison

Scenario: Use AI to create a product recommendation system

Copilot Prompt Use

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

Code :

```python
def quicksort(arr):
    """Sort an array using the QuickSort algorithm."""
    if len(arr) <= 1:
        return arr  # Base case: arrays with 0 or 1 element are already sorted
    else:
        pivot = arr[len(arr) // 2]  # Choose the pivot element
        left = [x for x in arr if x < pivot]  # Elements less than the pivot
        middle = [x for x in arr if x == pivot]  # Elements equal to the pivot
        right = [x for x in arr if x > pivot]  # Elements greater than the pivot
        # Recursively apply quicksort to left and right, and combine results
        return quicksort(left) + middle + quicksort(right)
def bubblesort(arr):    "bubblesort": Unknown word.
    """Sort an array using the BubbleSort algorithm."""
    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
if __name__ == "__main__":
    sample_array = [64, 34, 25, 12, 22, 11, 90]
    print("Original array:", sample_array)

    sorted_quick = quicksort(sample_array.copy())
    print("Sorted array using QuickSort:", sorted_quick)

    sorted_bubble = bubblesort(sample_array.copy())    "bubblesort": Unknown word.
    print("Sorted array using BubbleSort:", sorted_bubble)
```

Output :

```
PS D:\AI-Assisted coding> python -u "d:\AI-Assisted coding\Assingment5.1\Task4.py"
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]
PS D:\AI-Assisted coding>
```

Explanation:

QuickSort works by selecting a 'pivot' element and partitioning the other elements into two sub-arrays according to whether they are less than or greater than the pivot.

Task 5: Transparency in AI Recommendations

Scenario: Use AI to create a product recommendation system.

Copilot Prompt Use:

```
"""Generate a recommendation system that also provides reasons for each
suggestion." """
```

Code :

```python
def recommend_movies(user_preferences, movie_database):
    """Recommend movies based on user preferences."""
    recommendations = []
    reasons = []
    for movie in movie_database:
        score = 0
        reason = []
        # Check genre preference
        if movie['genre'] in user_preferences['preferred_genres']:
            score += 2
            reason.append(f"Matches preferred genre: {movie['genre']}")
        # Check director preference
        if movie['director'] in user_preferences['preferred_directors']:
            score += 1
            reason.append(f"Directed by preferred director: {movie['director']}")
        # Check actor preference
        if any(actor in user_preferences['preferred_actors'] for actor in movie['actors']):
            score += 1
            reason.append("Features preferred actors")
        # If the score is above a threshold, recommend the movie
        if score >= 2:
            recommendations.append(movie['title'])
            reasons.append((movie['title'], reason))
    return recommendations, reasons
if __name__ == "__main__":
    user_preferences = {
        'preferred_genres': ['Action', 'Sci-Fi'],
        'preferred_directors': ['Christopher Nolan'],
        'preferred_actors': ['Leonardo DiCaprio', 'Scarlett Johansson']    "Scarlett": Unknown word.
    }

    movie_database = [
        {
            'title': 'Inception',
            'genre': 'Sci-Fi',
            'director': 'Christopher Nolan',
            'actors': ['Leonardo DiCaprio', 'Joseph Gordon-Levitt']
        },
        {
            'title': 'The Dark Knight',
            'genre': 'Action',
            'director': 'Christopher Nolan',
            'actors': ['Christian Bale', 'Heath Ledger']
        },
        {
            'title': 'Lost in Translation',
```

```python
            'genre': 'Drama',
            'director': 'Sofia Coppola',
            'actors': ['Bill Murray', 'Scarlett Johansson']    "Scarlett": Unknown word.
        }
    ]
    recommendations, reasons = recommend_movies(user_preferences, movie_database)
    print("Recommended Movies:")
    for movie in recommendations:
        print(f"- {movie}")
    print("\nReasons for Recommendations:")
    for movie, reason in reasons:
        print(f"{movie}:")
        for r in reason:
            print(f"  - {r}")
```

Sample output:

```
PS D:\AI-Assisted coding> python -u "d:\AI-Assisted coding\Assingment5.1\Task5.py"
Recommended Movies:
- Inception
- The Dark Knight

Reasons for Recommendations:
Inception:
  - Matches preferred genre: Sci-Fi
  - Directed by preferred director: Christopher Nolan
  - Features preferred actors
The Dark Knight:
  - Matches preferred genre: Action
  - Directed by preferred director: Christopher Nolan
PS D:\AI-Assisted coding>
```

Explanation: The function recommend_movies takes user preferences and a movie database as input. It initializes empty lists for recommendations and reasons. It iterates through each movie in the database, calculating a score based on how well the movie matches user preferences. If the score meets a certain threshold, the movie is added to the recommendations list along the reasons for its recommendation. Finally, the main block demonstrates how to use the function with sample data and prints the recommendations along with their reasons.