AI-Assisted Assingment 5.1 Using GitHub Copilot

Name: Jayanth

Batch : 12

Rollno: 2303A51774

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 :

task1.py Assignment5 2 | task2.py Assignment5 | user_details.txt | task3.py Assignment5 | task4.py

Assignment5 > task1.py > ...

```python
# Ethical Foundations – Responsible AI Coding Practice# This code is developed with a focus on transparency, us
# Generate a Python program that fetches weather data from a public weather API. The code should be written sec
import os
import requests
from dotenv import load_dotenv
# Load environment variables from a .env file
from pathlib import Path
env_path = Path(__file__).parent / ".env"
load_dotenv(dotenv_path=env_path)
print("DEBUG API KEY:", os.getenv("WEATHER_API_KEY"))
API_KEY = os.getenv('WEATHER_API_KEY')  # Securely fetch the API key from environment variables
BASE_URL = "https://api.openweathermap.org/data/2.5/weather"
def get_weather(city_name):
    """Fetch weather data for a given city from the OpenWeatherMap API."""
    if not API_KEY:
        raise ValueError("API key is not set. Please set the WEATHER_API_KEY environment variable.")
    params = {
        'q': city_name,
        'appid': API_KEY,
        'units': 'metric'  # Fetch temperature in Celsius
    }
    response = requests.get(BASE_URL, params=params)
    if response.status_code == 200:
        return response.json()
    else:
        response.raise_for_status()
def display_weather_info(weather_data):
    """Display the fetched weather information."""
    city = weather_data['name']
    country = weather_data['sys']['country']
    temperature = weather_data['main']['temp']
    weather_description = weather_data['weather'][0]['description']
    print(f"Weather in {city}, {country}:")
    print(f"Temperature: {temperature}°C")
    print(f"Description: {weather_description.capitalize()}")
city_name = input("Enter city name: ")
try:
    weather_data = get_weather(city_name)
    display_weather_info(weather_data)
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 hashlib
import os
def hash_password(password):
    """Hash a password using SHA-256 to enhance security."""
    # Create a new sha256 hash object
    sha_signature = hashlib.sha256(password.encode()).hexdigest()
    return sha_signature
def save_user_details(name, email, hashed_password, filename='user_details.txt'):
    """Save user details to a file securely."""
    with open(filename, 'a') as file:
        file.write(f"Name: {name}, Email: {email}, Hashed Password: {hashed_password}\n")
# Collect user details
name = input("Enter your name: ")
email = input("Enter your email: ")
password = input("Enter your password: ")
# Hash the password before storing it
hashed_password = hash_password(password)
# Save the user details securely
save_user_details(name, email, hashed_password)
print("User details saved securely.")
# Hashing the password ensures that even if the file is accessed by unauthorized individuals,
```

Sample Output

```
Name: srikar, Email: nagavellisrikar@gmail.com, Hashed Password: 5772c02d4dd3e7804f9bb9ad06e04cc867d1c4285c5f5e2fc94b2232fc1df7f2
```

```
% /opt/homebrew/bin/python3 "/Users/srikarnagavelli/
Enter your name: srikar
Enter your email: nagavellisrikar@gmail.com
Enter your password: 1234
User details saved securely.
srikarnagavelli@Srikars-MacBook-Pro AI Assisted %
```

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

    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

    # 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

```
srikarnagavelli@Srikars-MacBook-Pro AI Assisted % /opt/homebr
li/Documents/AI Assisted/Assignment5/task3.py"
Enter a number to check if it is an Armstrong number: 123
123 is not an Armstrong number.
srikarnagavelli@Srikars-MacBook-Pro AI Assisted %
```

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 :

```
def bubble_sort(arr):
    """
    Sort an array using the Bubble Sort algorithm.

    Parameters:
    arr (list): The list of elements to be sorted.

    Returns:
    list: The sorted list.
    """
    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
def quick_sort(arr):
    """
    Sort an array using the Quick Sort algorithm.
    Parameters:
    arr (list): The list of elements to be sorted.
    """
    if len(arr) <= 1:
        return arr
    else:
        pivot = arr[len(arr) // 2]  # Choose the middle element as pivot
        left = [x for x in arr if x < pivot]  # Elements less than pivot
        middle = [x for x in arr if x == pivot]  # Elements equal to pivot
        right = [x for x in arr if x > pivot]  # Elements greater than pivot
        return quick_sort(left) + middle + quick_sort(right)  # Recursively sort and combine
try:
    user_input = input("Enter numbers to sort, separated by spaces: ")
    arr = list(map(int, user_input.split()))
    bubble_sorted_arr = bubble_sort(arr.copy())
    print("Bubble Sort Result:", bubble_sorted_arr)
    quick_sorted_arr = quick_sort(arr.copy())
    print("Quick Sort Result:", quick_sorted_arr)
except ValueError:
    print("Please enter valid integers separated by spaces.")
```

Output :

```
srikarnagavelli@Srikars-MacBook-Pro AI Assisted % /opt/homebre
li/Documents/AI Assisted/Assignment5/task4.py"
Enter numbers to sort, separated by spaces: 7 2 9 4 0 5
Bubble Sort Result: [0, 2, 4, 5, 7, 9]
Quick Sort Result: [0, 2, 4, 5, 7, 9]
srikarnagavelli@Srikars-MacBook-Pro AI Assisted %
```

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:

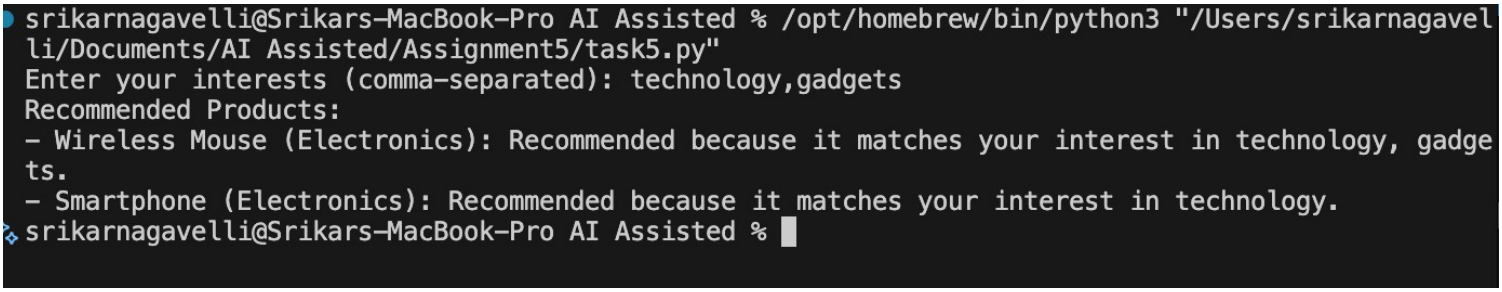
```
"""Generate a recommendation system that also provides reasons for each
suggestion." """
```

Code :

```
# Sample product database
products = {
    1: {"name": "Wireless Mouse", "category": "Electronics", "interests": ["technology", "gadgets"]},
    2: {"name": "Yoga Mat", "category": "Fitness", "interests": ["health", "wellness"]},
    3: {"name": "Cookbook", "category": "Books", "interests": ["cooking", "food"]},
    4: {"name": "Smartphone", "category": "Electronics", "interests": ["technology", "communication"]},
    5: {"name": "Running Shoes", "category": "Fitness", "interests": ["health", "sports"]},
    6: {"name": "Mystery Novel", "category": "Books", "interests": ["reading", "entertainment"]},
}
def recommend_products(user_interests):
    recommendations = []
    for product_id, product_info in products.items():
        # Safely retrieve interests to avoid KeyError
        product_interests = product_info.get("interests", [])
        # Check if user interests match product interests
        if any(interest in user_interests for interest in product_interests):
            explanation = (
                f"Recommended because it matches your interest in "
                f"{', '.join(set(user_interests).intersection(product_interests))}."
            )
            recommendations.append({
                "product": product_info["name"],
                "category": product_info["category"],
                "explanation": explanation
            })
    return recommendations
user_input = input("Enter your interests (comma-separated): ")
user_interests = [interest.strip().lower() for interest in user_input.split(",")]
recommended_products = recommend_products(user_interests)

if recommended_products:
    print("Recommended Products:")
    for rec in recommended_products:
        print(f"- {rec['product']} ({rec['category']}): {rec['explanation']}")
else:
    print("No recommendations available based on your interests.")
```

Sample output:



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.