AI Assisted Coding
Assignment - 5
V.Pranavith || 2303A51488 || Batch:- 8

Task Description #1 (Privacy in API Usage)

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

Code:

```
import os
import requests

# --- Configuration ---
# Define the name of the environment variable that stores your API key
API_KEY_ENV_VAR = 'YOUR_API_KEY_ENV_VAR_NAME' # e.g., 'OPENWEATHER_API_KEY'

# Define your weather API endpoint
WEATHER_API_ENDPOINT = 'YOUR_WEATHER_API_ENDPOINT' # e.g., 'https://api.openweathermap.org/data/2.5/weather'

# Define parameters for your API request (e.g., city, units)
params = {
    'q': 'London',
    'units': 'metric',
    # Add other parameters as required by your API
}

# --- Fetching API Key ---
try:
    api_key = os.environ[API_KEY_ENV_VAR]
    print(f"Successfully loaded API key from environment variable: {API_KEY_ENV_VAR}")
except KeyError:
    print(f"Error: Environment variable '{API_KEY_ENV_VAR}' not found.")
    print("Please set your API key as an environment variable. In Colab, use the 'Secrets' tab (🔑) on the left sidebar.")
    api_key = None

# --- Fetching Weather Data (if API key is available) ---
if api_key:
    # Add the API key to your parameters or headers as required by the API
    # For OpenWeatherMap, it's typically a 'appid' parameter
    params['appid'] = api_key

    try:
        response = requests.get(WEATHER_API_ENDPOINT, params=params)
        response.raise_for_status() # Raise an exception for HTTP errors

        weather_data = response.json()
        print("\n--- Weather Data ---")
        print(weather_data)

        # Example of accessing data
        # if 'main' in weather_data and 'temp' in weather_data['main']:
        #     print(f"Temperature in {params['q']}: {weather_data['main']['temp']}°C")

    except requests.exceptions.RequestException as e:
        print(f"Error fetching weather data: {e}")
    except ValueError as e:
        print(f"Error parsing JSON response: {e}")
else:
    print("Cannot fetch weather data without an API key.")
```

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.

Analyze: Check if the AI stores sensitive data in plain text or without encryption.

Code:

```
import csv

def store_user_data():
    name = input("Enter user name: ")
    email = input("Enter user email: ")
    password = input("Enter user password: ") # In a real application, NEVER store plain passwords.

    # Define the file name
    file_name = 'user_data.csv'

    # Check if the file already exists to decide whether to write headers
    file_exists = os.path.exists(file_name)

    try:
        with open(file_name, mode='a', newline='') as file:
            writer = csv.writer(file)

            if not file_exists:
                writer.writerow(['Name', 'Email', 'Password'])

            writer.writerow([name, email, password])
        print(f"User '{name}' data successfully stored in {file_name}")
    except IOError as e:
        print(f"Error writing to file: {e}")

# Call the function to store user data
store_user_data()
```

```
Enter user name: rohan
Enter user email: marrirohan18@gmail.com
Enter user password: rohan123
```

Task Description #3 (Transparency in Algorithm Design)
Objective: Use AI to generate an Armstrong number checking function with comments and explanations.

Code:

```
def is_armstrong(number):
    # Convert the number to a string to easily get its digits and count them
    num_str = str(number)
    num_digits = len(num_str)

    sum_of_powers = 0

    # Iterate through each digit of the number
    for digit_char in num_str:
        # Convert the character digit back to an integer
        digit = int(digit_char)
        # Add the digit raised to the power of the total number of digits
        sum_of_powers += digit ** num_digits

    # Compare the sum of powers with the original number
    return sum_of_powers == number

# --- Examples of usage ---
print(f"Is 9 an Armstrong number? {is_armstrong(9)}")        # Expected: True
print(f"Is 10 an Armstrong number? {is_armstrong(10)}")      # Expected: False
print(f"Is 153 an Armstrong number? {is_armstrong(153)}")    # Expected: True
print(f"Is 370 an Armstrong number? {is_armstrong(370)}")    # Expected: True
print(f"Is 371 an Armstrong number? {is_armstrong(371)}")    # Expected: True
print(f"Is 407 an Armstrong number? {is_armstrong(407)}")    # Expected: True
print(f"Is 1634 an Armstrong number? {is_armstrong(1634)}") # Expected: True
print(f"Is 1000 an Armstrong number? {is_armstrong(1000)}") # Expected: False
```

```
Is 9 an Armstrong number? True
Is 10 an Armstrong number? False
Is 153 an Armstrong number? True
Is 370 an Armstrong number? True
Is 371 an Armstrong number? True
Is 407 an Armstrong number? True
Is 1634 an Armstrong number? True
Is 1000 an Armstrong number? False
```

Task Description #4 (Transparency in Algorithm Comparison)
Task: Use AI to implement two sorting algorithms (e.g., QuickSort and BubbleSort).

Code:

```python
def bubble_sort(arr):
    n = len(arr)
    # Traverse through all array elements
    for i in range(n - 1):
        # last i elements are already in place, so we don't need to check them
        swapped = False # Optimization: if no two elements were swapped by inner loop, then break
        # Traverse the array from 0 to n-i-1
        # Compare adjacent elements and swap them if the current is greater than the next
        for j in range(n - 1 - i):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j] # Swap elements
                swapped = True
        # If no two elements were swapped by inner loop, then break
        if not swapped:
            break
    return arr


def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    else:
        # Choosing the last element as the pivot
        pivot = arr[-1]
        less = []
        greater = []
        equal = []

        # Partitioning step
        for x in arr:
            if x < pivot:
                less.append(x)
            elif x > pivot:
                greater.append(x)
            else:
                equal.append(x)

        # Recursively sort the sub-arrays and combine
        return quick_sort(less) + equal + quick_sort(greater)

# --- Example Usage ---
my_list_bubble = [64, 34, 25, 12, 22, 11, 90]
print(f"Original list for Bubble Sort: {my_list_bubble}")
print(f"Sorted list (Bubble Sort): {bubble_sort(list(my_list_bubble))}\n") # Use list() to pass a copy

my_list_quick = [64, 34, 25, 12, 22, 11, 90]
print(f"Original list for QuickSort: {my_list_quick}")
print(f"Sorted list (QuickSort): {quick_sort(list(my_list_quick))}") # Use list() to pass a copy
```

```
Original list for Bubble Sort: [64, 34, 25, 12, 22, 11, 90]
Sorted list (Bubble Sort): [11, 12, 22, 25, 34, 64, 90]

Original list for QuickSort: [64, 34, 25, 12, 22, 11, 90]
Sorted list (QuickSort): [11, 12, 22, 25, 34, 64, 90]
```

Task Description #5 (Transparency in AI Recommendations)

Task: Use AI to create a product recommendation system.

Code:

```python
import pandas as pd
movies_data = [
    {
        'movie_id': 1,
        'title': 'The Shawshank Redemption',
        'genres': 'Drama',
        'keywords': 'prison|escape|friendship|hope',
        'director': 'Frank Darabont',
        'cast': 'Tim Robbins|Morgan Freeman',
        'overview': 'Two imprisoned men bond over a number of years, finding solace and eventual redemption through acts of common decency.',
        'release_year': 1994,
        'vote_average': 8.7
    },
    {
        'movie_id': 2,
        'title': 'The Dark Knight',
        'genres': 'Action|Crime|Drama',
        'keywords': 'gotham|superhero|villain|psychopath',
        'director': 'Christopher Nolan',
        'cast': 'Christian Bale|Heath Ledger|Aaron Eckhart',
        'overview': 'When the menace known as The Joker wreaks havoc and chaos on the people of Gotham, Batman must accept one of the greatest psychological
        'release_year': 2008,
        'vote_average': 8.5
    },
    {
        'movie_id': 3,
        'title': 'Pulp Fiction',
        'genres': 'Crime|Drama',
        'keywords': 'hitman|gangster|nonlinear storytelling|bloody',
        'director': 'Quentin Tarantino',
        'cast': 'John Travolta|Samuel L. Jackson|Uma Thurman',
        'overview': 'The lives of two mob hitmen, a boxer, a gangster and his wife, and a pair of diner bandits intertwine in four tales of violence and red
        'release_year': 1994,
        'vote_average': 8.4

    }
]

# 3. Convert the list of movie dictionaries into a pandas DataFrame
movies_df = pd.DataFrame(movies_data)

# 4. Create another list of dictionaries for user preference data
user_preferences_data = [
    {'user_id': 1, 'movie_id': 1, 'rating': 5},
    {'user_id': 1, 'movie_id': 2, 'rating': 4},
    {'user_id': 1, 'movie_id': 3, 'rating': 5},
    {'user_id': 2, 'movie_id': 1, 'rating': 4},
    {'user_id': 2, 'movie_id': 4, 'rating': 3},
    {'user_id': 2, 'movie_id': 5, 'rating': 5},
    {'user_id': 3, 'movie_id': 2, 'rating': 5},
    {'user_id': 3, 'movie_id': 3, 'rating': 4},
    {'user_id': 3, 'movie_id': 6, 'rating': 4}
]

# 5. Convert the list of user preference dictionaries into a pandas DataFrame
user_preferences_df = pd.DataFrame(user_preferences_data)

# 6. Display the first few rows and the info() for both DataFrames
print("--- movies_df ---")
print(movies_df.head())
print("\n")
movies_df.info()

print("\n--- user_preferences_df ---")
print(user_preferences_df.head())
print("\n")
user_preferences_df.info()
```

```
--- movies_df ---
   movie_id                    title             genres  \
0         1  The Shawshank Redemption              Drama
1         2           The Dark Knight  Action|Crime|Drama
2         3              Pulp Fiction         Crime|Drama
```