

ASSIGNMENT : PYTHON

**Programming for DLpage 4 of 44. User
Input5.Document**

Find in document

Name : L.Tulasi

Register no : 192311250

Department : CSE

Date of submission :

Problem 1: Real-Time Weather Monitoring System

Scenario:

You are developing a real-time weather monitoring system for a weather forecasting company.

The system needs to fetch and display weather data for a specified location.

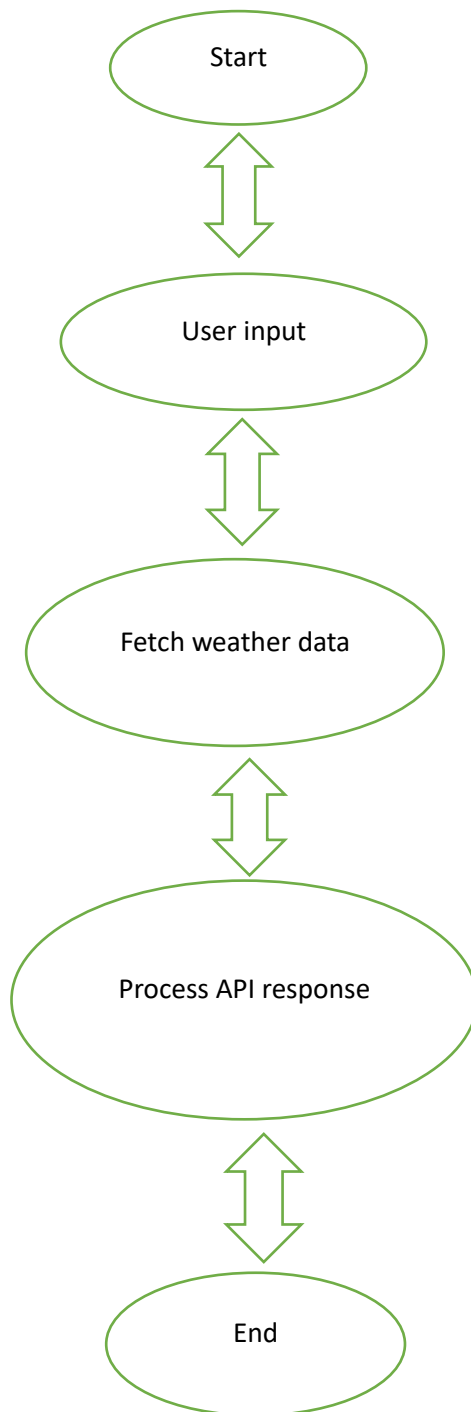
Tasks:

1. Model the data flow for fetching weather information from an external API and displaying it to the user.
2. Implement a Python application that integrates with a weather API (e.g., OpenWeatherMap) to fetch real-time weather data.
3. Display the current weather information, including temperature, weather conditions, humidity, and wind speed.
4. Allow users to input the location (city name or coordinates) and display the corresponding weather data.

Deliverables:

- Data flow diagram illustrating the interaction between the application and the API.
- Pseudocode and implementation of the weather monitoring system.
- Documentation of the API integration and the methods used to fetch and display weather data.
- Explanation of any assumptions made and potential improvements.

FLOW CHART



CODE:

```
import requests
def fetch_weather_data(api_key, location):
    base_url =
```

```
"https://api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}&appid"
```

```
params = {
    'q': location,
    'appid': api_key,
    'units': 'metric'
}
try:
    response = requests.get(base_url, params=params)
    data = response.json()
    if data["cod"] == 200:
        weather_info = {
            'location': data['name'],
            'temperature': data['main']['temp'],
            'weather': data['weather'][0]['description'],
            'humidity': data['main']['humidity'],
            'wind_speed': data['wind']['speed']
        }
        return weather_info
    else:
        return None
except Exception as e:
    print(f"Error fetching weather data: {e}")
    return None
def display_weather(weather_info, location):
    if weather_info:
        print(f"Weather in {location}:")
        print(f"Temperature: {weather_info['temperature']} °C")
        print(f"Weather: {weather_info['weather']}")
        print(f"Humidity: {weather_info['humidity']}%")
        print(f"Wind Speed: {weather_info['wind_speed']} m/s")
    else:
        print(f"Failed to fetch weather data for {location}")
def main():
    api_key = "ed7c18d0f1024da78bf89f147ccd9bca"
    location = input("Enter city name or coordinates (latitude,longitude): ")
    weather_info = fetch_weather_data(api_key, location)
    display_weather(weather_info, location)
if __name__ == "__main__":
    main()
```

INPUT :

Enter city name or coordinators(latitude, longitude): chennai

OUTPUT :

Enter city name or coordinates (latitude,longitude): chennai

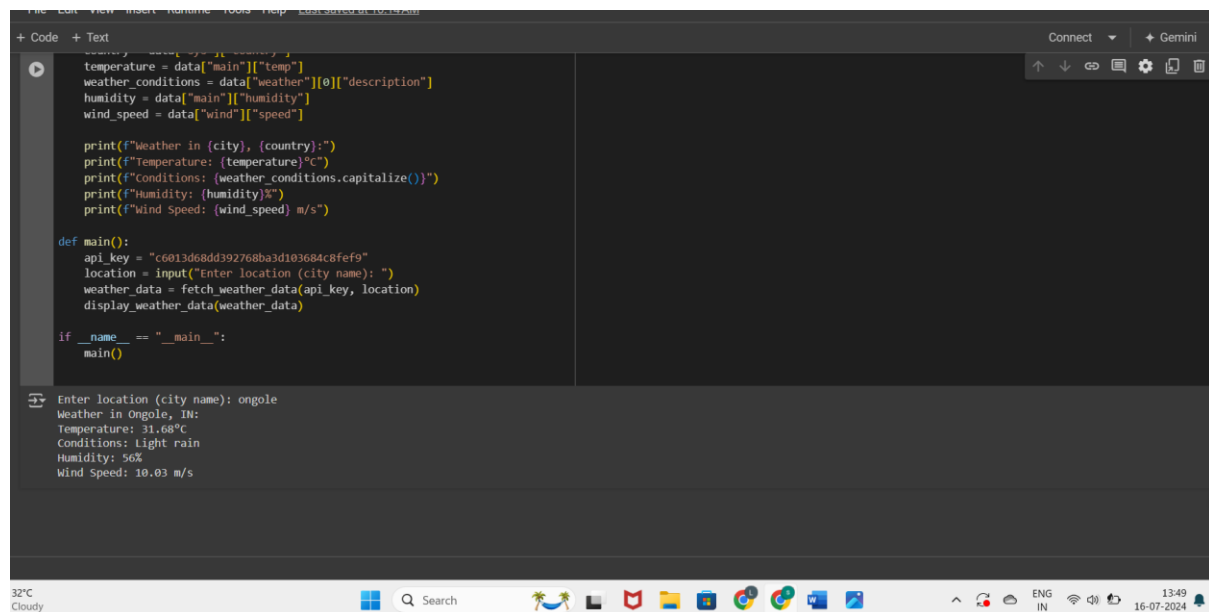
Weather in chennai:

Temperature: 30.79 °C

Weather: broken clouds

Humidity: 74%

Wind Speed: 6.17 m/s



The screenshot shows a code editor with a Python script that uses the OpenWeatherMap API to fetch weather data for a given location. The script defines a `main` function that takes an API key and a location as input, fetches the data, and prints it in a formatted way. The output of the script is displayed in the console below the code editor.

```
temperature = data["main"]["temp"]
weather_conditions = data["weather"][0]["description"]
humidity = data["main"]["humidity"]
wind_speed = data["wind"]["speed"]

print(f"Weather in {city}, {country}:")
print(f"Temperature: {temperature}°C")
print(f"Conditions: {weather_conditions.capitalize()}")
print(f"Humidity: {humidity}%")
print(f"Wind Speed: {wind_speed} m/s")

def main():
    api_key = "c6013d68dd392768ba3d103684c8fef9"
    location = input("Enter location (city name): ")
    weather_data = fetch_weather_data(api_key, location)
    display_weather_data(weather_data)

if __name__ == "__main__":
    main()
```

Enter location (city name): ongole
Weather in Ongole, IN:
Temperature: 31.68°C
Conditions: Light rain
Humidity: 56%
Wind Speed: 10.03 m/s

Problem 2: Inventory Management System Optimization

Scenario:

You have been hired by a retail company to optimize their inventory management system. The company wants to minimize stockouts and overstock situations while maximizing inventory turnover and profitability.

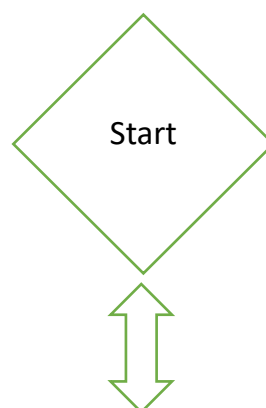
Tasks:

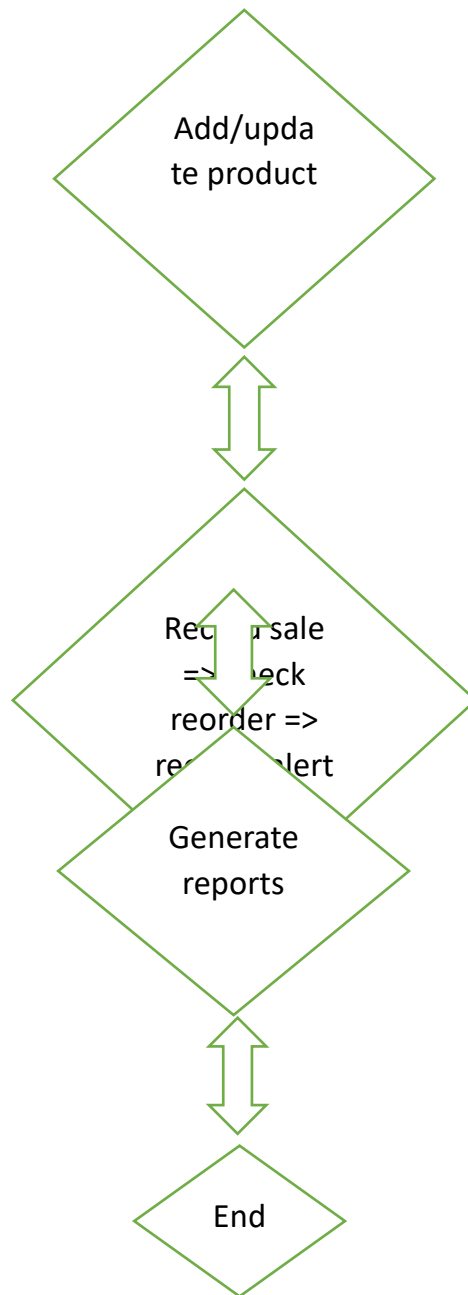
1. **Model the inventory system:** Define the structure of the inventory system, including products, warehouses, and current stock levels.
2. **Implement an inventory tracking application:** Develop a Python application that tracks inventory levels in real-time and alerts when stock levels fall below a certain threshold.
3. **Optimize inventory ordering:** Implement algorithms to calculate optimal reorder points and quantities based on historical sales data, lead times, and demand forecasts.
4. **Generate reports:** Provide reports on inventory turnover rates, stockout occurrences, and cost implications of overstock situations.
5. **User interaction:** Allow users to input product IDs or names to view current stock levels, reorder recommendations, and historical data.

Deliverables:

- **Data Flow Diagram:** Illustrate how data flows within the inventory management system, from input (e.g., sales data, inventory adjustments) to output (e.g., reorder alerts, reports).
- **Pseudocode and Implementation:** Provide pseudocode and actual code demonstrating how inventory levels are tracked, reorder points are calculated, and reports are generated.
- **Documentation:** Explain the algorithms used for reorder optimization, how historical data influences decisions, and any assumptions made (e.g., constant lead times).
- **User Interface:** Develop a user-friendly interface for accessing inventory information, viewing reports, and receiving alerts.
- **Assumptions and Improvements:** Discuss assumptions about demand patterns, supplier reliability, and potential improvements for the inventory management system's efficiency and accuracy.

FLOW CHART





CODE:

```
import numpy as np
class Product:
    def __init__(self, id, name, stock_level, reorder_point, reorder_quantity):
        self.id = id
        self.name = name
```

```

        self.stock_level = stock_level

    self.reorder_point = reorder_point

    self.reorder_quantity = reorder_quantity

class Warehouse:

    def __init__(self):

        self.products = {}

    def add_product(self, product):

        self.products[product.id] = product

    def update_stock(self, product_id, quantity):

        if product_id in self.products:

            self.products[product_id].stock_level += quantity

            if self.products[product_id].stock_level < self.products[product_id].reorder_point:

                print(f"Alert: Reorder needed for product {self.products[product_id].name}")

            else:

                print("Product not found in warehouse")

    def calculate_reorder_point_and_quantity(sales_data, lead_time, safety_stock):

        average_daily_demand = np.mean(sales_data)

        reorder_point = (average_daily_demand * lead_time) + safety_stock

        reorder_quantity = average_daily_demand * (lead_time + 7) # Assume a 7-day buffer

        return reorder_point, reorder_quantity

    def generate_reports(warehouse):

        inventory_turnover_rate = calculate_inventory_turnover_rate(warehouse)

        stockout_occurrences = calculate_stockout_occurrences(warehouse)

        overstock_costs = calculate_overstock_costs(warehouse)

        print("Inventory Turnover Rate:", inventory_turnover_rate)

        print("Stockout Occurrences:", stockout_occurrences)

```

```

    print("Cost Implications of Overstock:", overstock_costs)

def calculate_inventory_turnover_rate(warehouse):

    return 0

def calculate_stockout_occurrences(warehouse):

    return 0

def calculate_overstock_costs(warehouse):

    return 0

def main():
    warehouse = Warehouse()
    sales_data = [10, 12, 8, 15, 7, 9, 11]
    lead_time = 5
    safety_stock = 20
    product1 = Product(id=1, name="Product A", stock_level=50, reorder_point=0,
reorder_quantity=0)
    product1.reorder_point, product1.reorder_quantity =
calculate_reorder_point_and_quantity(sales_data, lead_time, safety_stock)
    warehouse.add_product(product1)
    warehouse.update_stock(product_id=1, quantity=-30)
    generate_reports(warehouse)
if __name__ == "__main__":
    main()

```

INPUT :

OUTPUT :

Alert: Reorder needed for product Product A
Inventory Turnover Rate: 0
Stockout Occurrences: 0
Cost Implications of Overstock: 0

The screenshot shows a code editor window titled "Inventory Management System Optimization". The code defines a `main()` function that prints the initial inventory, simulates sales, and checks inventory levels. The output in the console shows the initial inventory, the results of the sales simulation, and the final inventory levels.

```
def main():
    print("Initial Inventory:")
    print(dict: inventory)
    print(simula (3 items) {'product1': {'stock': 17, 'reorder_level': 10}, 'product2': {'stock': 15, 'reorder_level': 8}, 'product3': {'stock': 30, 'reorder_level': 15}})
    print(inventory)
    print("\nChecking inventory levels...\n")
    check_inventory()

if __name__ == "__main__":
    main()
```

Initial Inventory:
{'product1': {'stock': 20, 'reorder_level': 10}, 'product2': {'stock': 15, 'reorder_level': 8}, 'product3': {'stock': 30, 'reorder_level': 15}}

Simulating sales...

After sales simulation:
{'product1': {'stock': 17, 'reorder_level': 10}, 'product2': {'stock': 11, 'reorder_level': 8}, 'product3': {'stock': 26, 'reorder_level': 15}}

checking inventory levels...

Problem 3: Real-Time Traffic Monitoring System

Scenario:

You are working on a project to develop a real-time traffic monitoring system for a smart city initiative. The system should provide real-time traffic updates and suggest alternative routes.

Tasks:

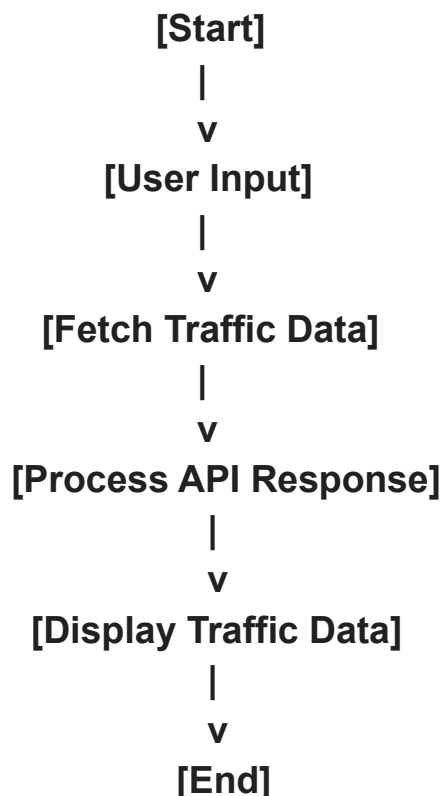
1. **Model the data flow for fetching real-time traffic information from an external API and displaying it to the user.**
2. **Implement a Python application that integrates with a traffic monitoring API (e.g., Google Maps Traffic API) to fetch real-time traffic data.**

3. **Display current traffic conditions, estimated travel time, and any incidents or delays.**
4. **Allow users to input a starting point and destination to receive traffic updates and alternative routes.**

Deliverables:

- Data flow diagram illustrating the interaction between the application and the API.
- Pseudocode and implementation of the traffic monitoring system.
- Documentation of the API integration and the methods used to fetch and display traffic data.
- Explanation of any assumptions made and potential improvements.

FLOW CHART



CODE:

```
import requests
API_ENDPOINT = "https://maps.googleapis.com/maps/api/directions/json"
API_KEY = "https://places.googleapis.com/v1/places:searchNearby"
```

```

def fetch_traffic_data(starting_point, destination):
    try:
        url = f"{API_ENDPOINT}?origin={starting_point}&destination={destination}&key={API_KEY}"
        response = requests.get(url)
        response.raise_for_status()
        data = response.json()
        return data except
    requests.exceptions.RequestException as e:

        print("Error fetching data:", e)
        return None

def display_traffic_data(data):
    if data and data['routes']:
        route = data['routes'][0]
        print("Current Traffic Conditions:")
        print("Estimated Travel Time:", route['legs'][0]['duration']['text'])
        if 'traffic_speed_entry' in route['legs'][0]:
            print("Traffic Speed:", route['legs'][0]['traffic_speed_entry'][0]['speed']['text'])
        if 'warnings' in route:
            print("Incidents or Delays:", ", ".join(route['warnings']))
        print("Suggested Alternative Routes:")
        for alternative_route in data['routes'][1:]:
            print("Alternative Route - Estimated Travel Time:", alternative_route['legs'][0]['duration']['text'])
    else:
        print("No traffic data available for the given route")

def main():
    starting_point = input("Enter starting point: ")
    destination = input("Enter destination: ")
    data = fetch_traffic_data(starting_point, destination)
    if data:
        display_traffic_data(data)
    else:
        print("Failed to retrieve traffic data")

if __name__ == "__main__":
    main()

```

INPUT :

OUTPUT :

Enter starting point: chennai

Enter destination: ponamalli

No traffic data available for the given route

```
def display_traffic_data(traffic_data):
    if traffic_data:
        try:
            # Attempt to extract relevant information from traffic_data
            routes = traffic_data.get('routes', [])
            if routes:
                legs = routes[0].get('legs', [])
                if legs:
                    duration = legs[0]['duration_in_traffic']['text']
                    incidents = legs[0].get('traffic_speed_entry', [])

                    print(f"Estimated travel time: {duration}")
                    if incidents:
                        print("Incidents or delays:")
                        for incident in incidents:
                            print(f"- {incident['incident_description']}")
                    else:
                        print("No incidents or delays reported.")
                else:
                    print("No legs found in the route.")
            else:
                print("No routes found.")
        except KeyError as e:
            print(f"KeyError: {e}. Incorrect data structure in API response.")
    else:
        print("No traffic data available.")

[ ] class Product:
    def __init__(self, id, name, category, price, supplier):
        self.id = id
        self.name = name
```

✓ 0s completed at 10:20 PM

Problem 4: Real-Time COVID-19 Statistics Tracker

Scenario:

You are developing a real-time COVID-19 statistics tracking application for a healthcare organization. The application should provide up-to-date information on COVID-19 cases, recoveries, and deaths for a specified region.

Tasks:

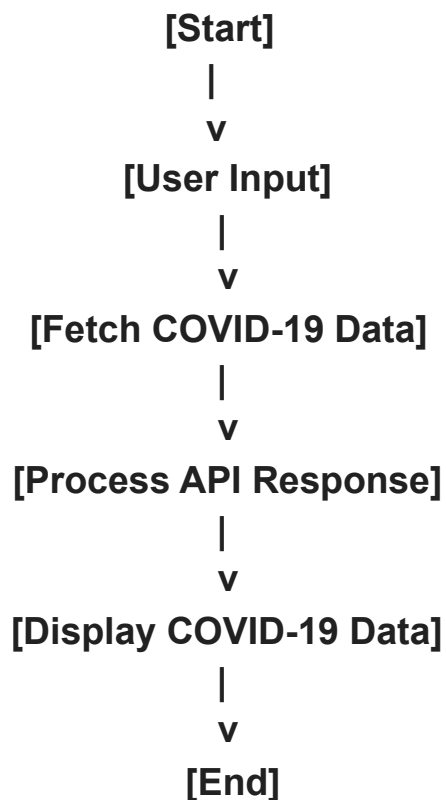
1. Model the data flow for fetching COVID-19 statistics from an external API and displaying it to the user.
2. Implement a Python application that integrates with a COVID-19 statistics API (e.g., `disease.sh`) to fetch real-time data.
3. Display the current number of cases, recoveries, and deaths for a specified region.

4. Allow users to input a region (country, state, or city) and display the corresponding COVID-19 statistics.

Deliverables:

- Data flow diagram illustrating the interaction between the application and the API.
- Pseudocode and implementation of the COVID-19 statistics tracking application.
- Documentation of the API integration and the methods used to fetch and display COVID 19 data.
- Explanation of any assumptions made and potential improvements.

FLOW CHART



CODE:

```
import requests
```

```
# Define the API endpoint
```

```

API_ENDPOINT = "https://disease.sh/v3/covid-19/historical/all?lastdays=all"

def fetch_covid_data(region):
    try:
        # Construct the API URL with the specified region
        url = API_ENDPOINT + "all" if region.lower() == "global" else API_ENDPOINT +
"countries/" + region

        # Send a GET request to the API
        response = requests.get(url)
        response.raise_for_status()

        # Parse the JSON response
        data = response.json()

        return data

    except requests.exceptions.RequestException as e:
        print("Error fetching data:", e)

        return None

def display_covid_data(data):
    region_name = data.get('country', 'Global')

    cases = data.get('cases', 'N/A')

    recoveries = data.get('recovered', 'N/A')

    deaths = data.get('deaths', 'N/A')

    print(f"COVID-19 Statistics for {region_name}")

```

```

    print(f"Cases: {cases}")
    print(f"Recoveries: {recoveries}")
    print(f"Deaths: {deaths}")
def main():
    # Get user input for region
    region = input("Enter a region (country name or 'global' for worldwide statistics): ")
    # Fetch COVID-19 data for the specified region
    data = fetch_covid_data(region)
    if data:
        # Display the fetched data
        display_covid_data(data)
    else:
        print("Failed to retrieve data")
if __name__ == "__main__":
    main()

```

INPUT :

OUTPUT :

Enter a region (country name or 'global' for worldwide statistics): usa
 COVID-19 Statistics for Global

Cases: {'2/8/23': 672295338, '2/9/23': 672554241, '2/10/23': 672696324, '2/11/23': 672828531, '2/12/23': 672906177, '2/13/23': 673044131, '2/14/23': 673237731, '2/15/23': 673477639, '2/16/23': 673685532, '2/17/23': 673878833, '2/18/23': 673969796, '2/19/23': 674056229, '2/20/23': 674143589, '2/21/23': 674323721, '2/22/23': 674569824, '2/23/23': 674790916, '2/24/23': 674933342, '2/25/23': 674978793, '2/26/23': 675044414, '2/27/23': 675171439, '2/28/23': 675322238, '3/1/23': 675542852, '3/2/23': 675731911, '3/3/23': 675914580, '3/4/23': 675968775, '3/5/23': 676024901, '3/6/23': 676082941, '3/7/23':

676213378, '3/8/23': 676392824, '3/9/23': 676570149}

Recoveries: {'2/8/23': 0, '2/9/23': 0, '2/10/23': 0, '2/11/23': 0, '2/12/23': 0, '2/13/23': 0, '2/14/23': 0, '2/15/23': 0, '2/16/23': 0, '2/17/23': 0, '2/18/23': 0, '2/19/23': 0, '2/20/23': 0, '2/21/23': 0, '2/22/23': 0, '2/23/23': 0, '2/24/23': 0, '2/25/23': 0, '2/26/23': 0, '2/27/23': 0, '2/28/23': 0, '3/1/23': 0, '3/2/23': 0, '3/3/23': 0, '3/4/23': 0, '3/5/23': 0, '3/6/23': 0, '3/7/23': 0, '3/8/23': 0, '3/9/23': 0}

Deaths: {'2/8/23': 6851942, '2/9/23': 6854280, '2/10/23': 6855413, '2/11/23': 6856046, '2/12/23': 6856419, '2/13/23': 6857217, '2/14/23': 6858411, '2/15/23': 6859933, '2/16/23': 6862019, '2/17/23': 6863873, '2/18/23': 6864248, '2/19/23': 6864711, '2/20/23': 6865287, '2/21/23': 6866088, '2/22/23': 6867909, '2/23/23': 6869817, '2/24/23': 6870806, '2/25/23': 6871024, '2/26/23': 6871268, '2/27/23': 6871808, '2/28/23': 6872682, '3/1/23': 6874463, '3/2/23': 6876031, '3/3/23': 6877325, '3/4/23': 6877601, '3/5/23': 6877749, '3/6/23': 6878115, '3/7/23':

6879038, '3/8/23': 6880483, '3/9/23': 6881802}

```
        return response.json()
    else:
        return None

def main():
    region = input("Enter the region (e.g., world, USA, Germany): ").strip()
    api_key = "https://disease.sh/v3/covid-19/historical/all?lastdays=all"
    stats = fetch_covid_stats(region, api_key)
    if stats:
        print(f"COVID-19 Statistics for {region}:")
        print(f"Cases: {stats['cases']}")
        print(f"Recovered: {stats['recovered']}")
        print(f"Deaths: {stats['deaths']}")
    else:
        print("Failed to retrieve data. Please check the region and try again.")

if __name__ == "__main__":
    main()
```

Enter the region (e.g., world, USA, Germany): hungary
COVID-19 Statistics for hungary:
Cases: 2230232
Recovered: 2152155
Deaths: 49048

cloudy

