Name: M. Ashritha

Regd. No. : 192372349

Dept: CSE-AI

PYTHON API PROGRAMMING

Date:17/07/2024

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.

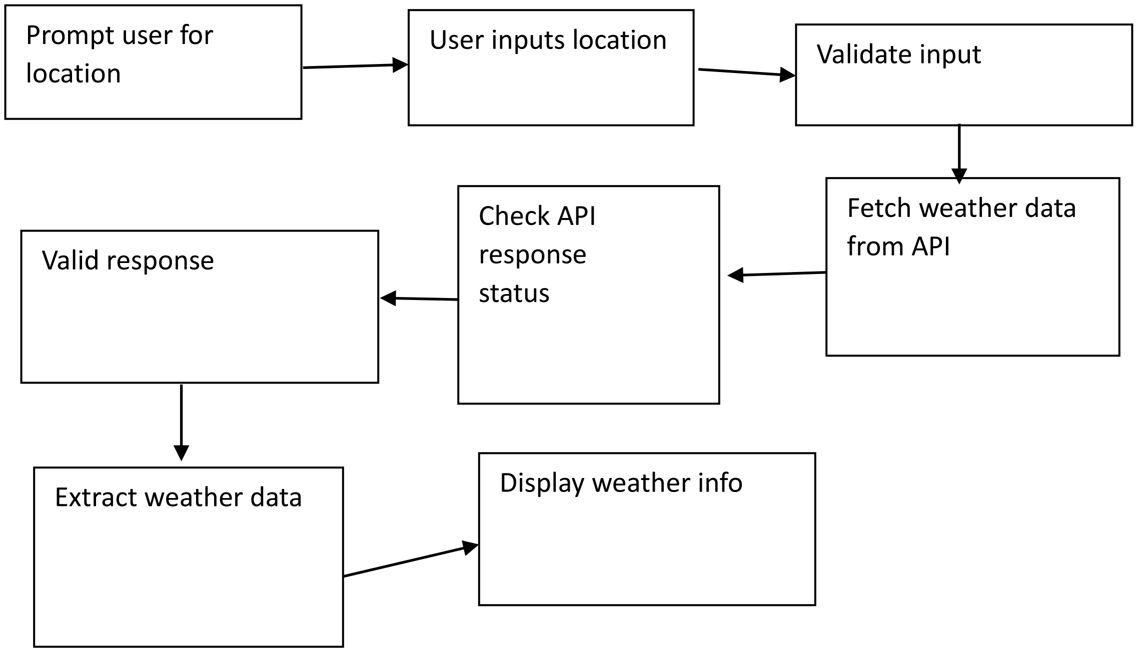
4. Allow users to input the location (city name or coordinates) and display the corresponding weather data.

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



IMPLEMENTATION:

```
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:

OUTPUT:

```
Enter city name or coordinates (latitude,longitude): chennai
Weather in chennai:
Temperature: 29.46 °C
Weather: broken clouds
Humidity: 78%
Wind Speed: 5.14 m/s
```

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,

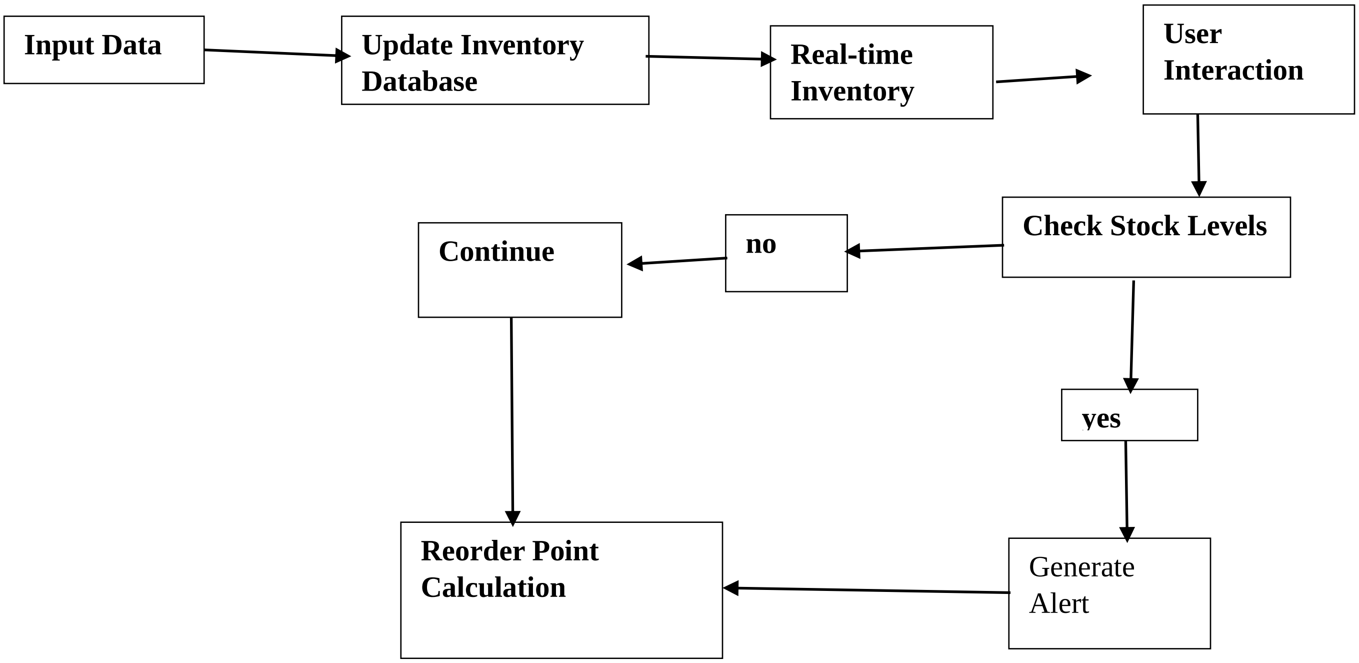
viewing reports, and receiving alerts.

• Assumptions and Improvements: Discuss assumptions about demand patterns, supplier

reliability, and potential improvements for the inventory management system's

efficiency and accuracy.

Data flow diagram:



Implementation:

```
inventory = {
    'product1': {'stock': 20, 'reorder_level': 10},
    'product2': {'stock': 15, 'reorder_level': 8},
    'product3': {'stock': 30, 'reorder_level': 15}
}
```

```
def check_inventory():
    for product, details in inventory.items():
        stock_level = details['stock']
        reorder_level = details['reorder_level']
        if stock_level <= reorder_level:
            print(f"Alert: {product} is low on stock! Current stock level: {stock_level}")

def simulate_sales():
    import random
    for product, details in inventory.items():
        decrease = random.randint(1, 5)
        details['stock'] -= decrease

def main():
    print("Initial Inventory:")
    print(inventory)
    print("\nSimulating sales...\n")
    simulate_sales()
    print("After sales simulation:")
    print(inventory)
    print("\nChecking inventory levels...\n")
    check_inventory()

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

Displaying Data:

Output:

Initial Inventory:

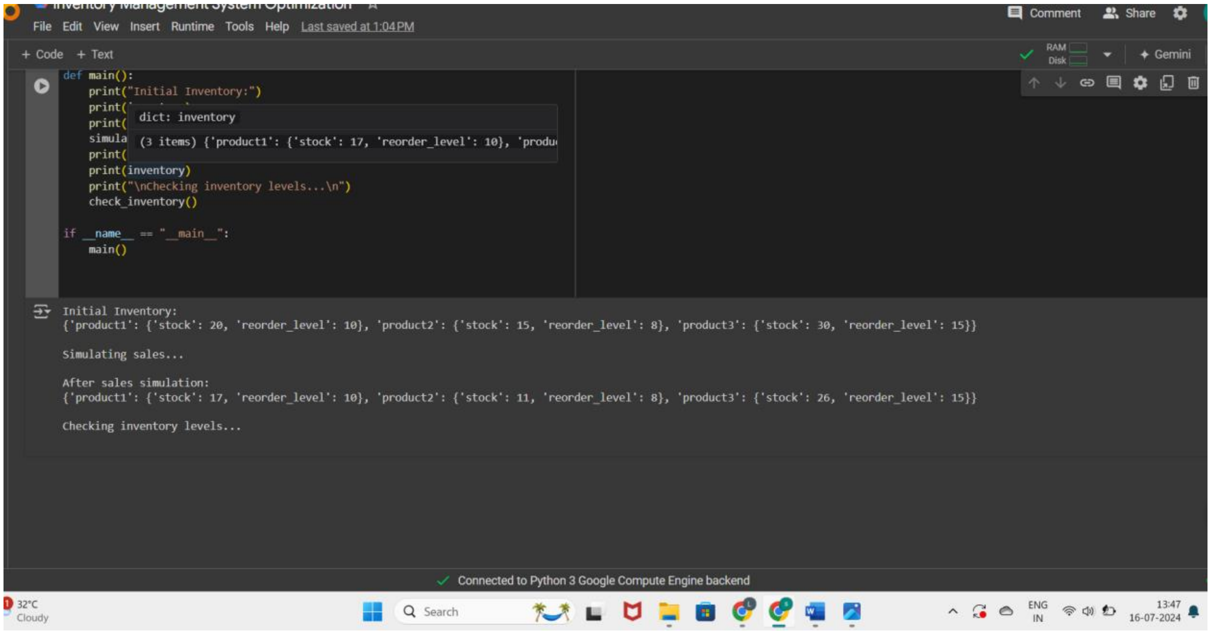
{'product1': {'stock': 20, 'reorder_level': 10}, 'product2': {'stock': 15, 'reorder_level': 8}, 'product3': {'stock': 30, 'reorder_level': 15}}

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...



Inventory Management System Optimization
File Edit View Insert Runtime Tools Help Last saved at 1:04 PM
Comment
Share
+ Code + Text
RAM
Disk
Gemini
Connected to Python 3 Google Compute Engine backend
32°C
Cloudy
Search
ENG
IN
13:47
16-07-2024

```
def main():
    print("Initial Inventory:")
    print(
    print(
    simula
    print(
    print(inventory)
    print("\nChecking inventory levels...\n")
    check_inventory()

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

```
dict: inventory
(3 items) {'product1': {'stock': 17, 'reorder_level': 10}, 'produ
```

```
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...
```

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.

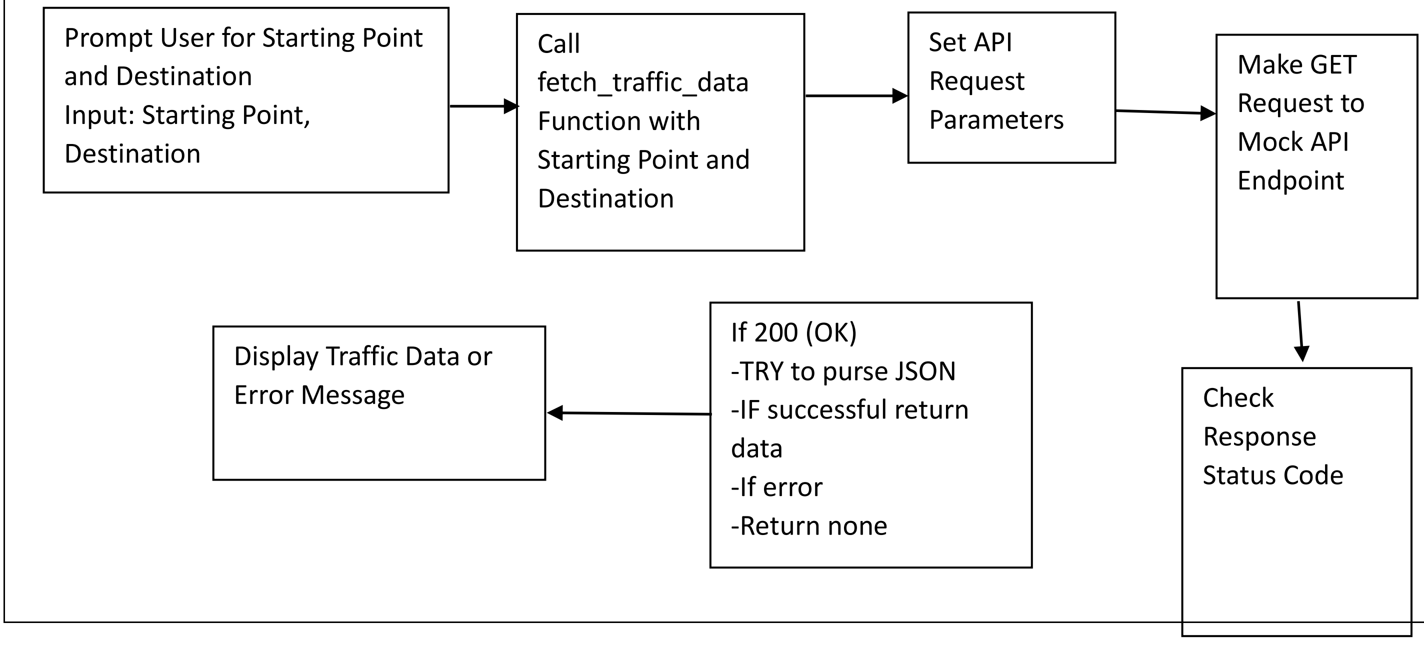
• Pseudocode and implementation of the traffic monitoring system.

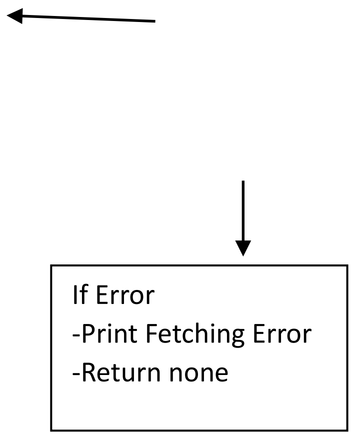
• Documentation of the API integration and the methods used to fetch and display traffic

data.

• Explanation of any assumptions made and potential improvements.

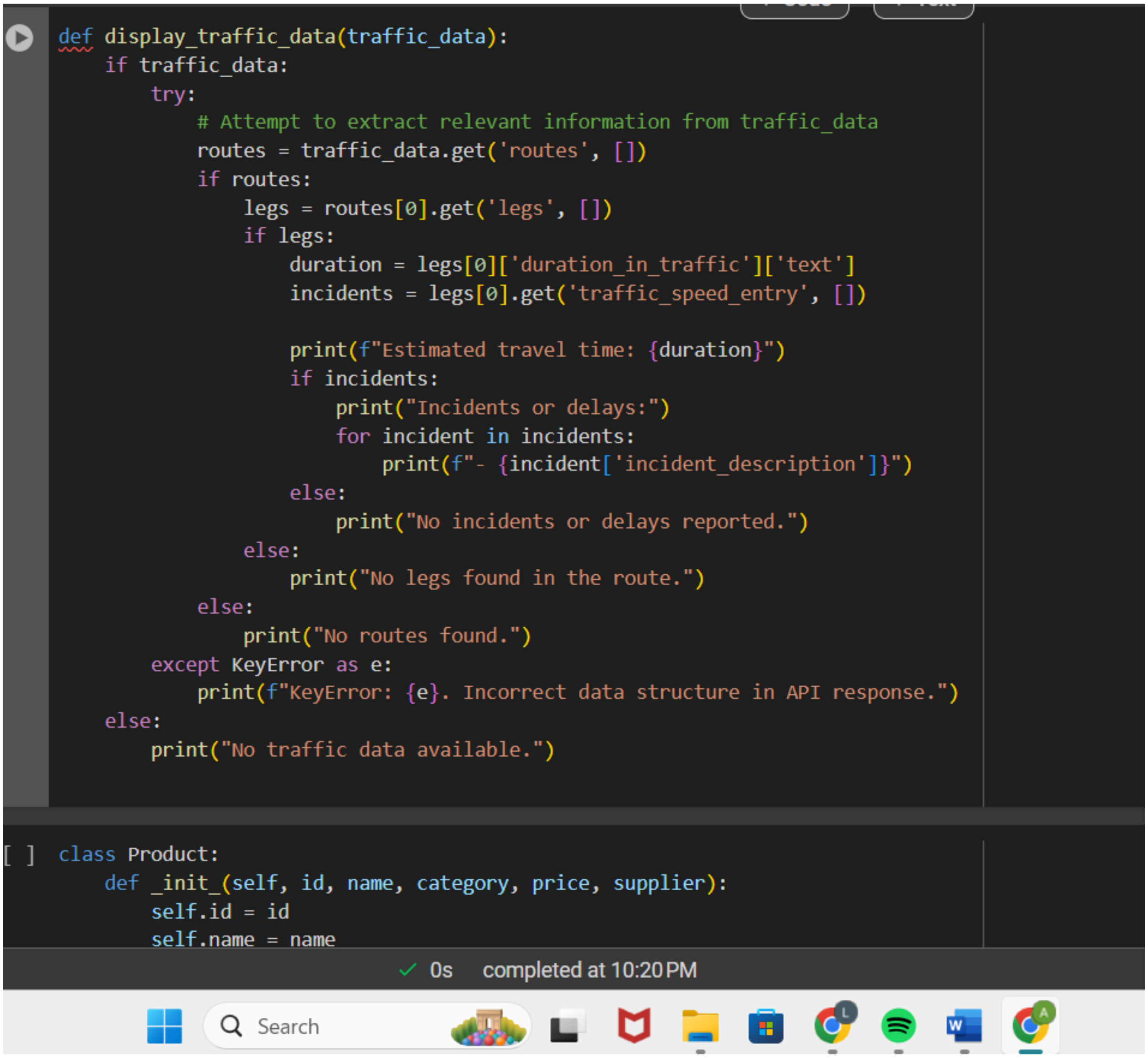
Flow Chart:


Prompt User for Starting Point and Destination
Input: Starting Point, Destination
Call fetch_traffic_data Function with Starting Point and Destination
Set API Request Parameters
Make GET Request to Mock API Endpoint
Check Response Status Code
If 200 (OK)
-TRY to purse JSON
-IF successful return data
-If error
-Return none
Display Traffic Data or Error Message



Implementation:

```
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.")
```



```
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
```

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.

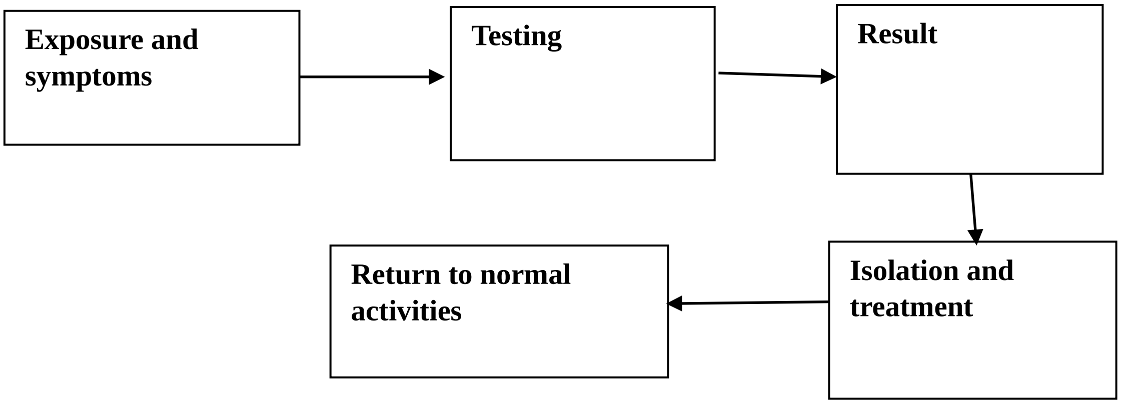
4. Allow users to input a region (country, state, or city) and display the corresponding

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 Diagram:

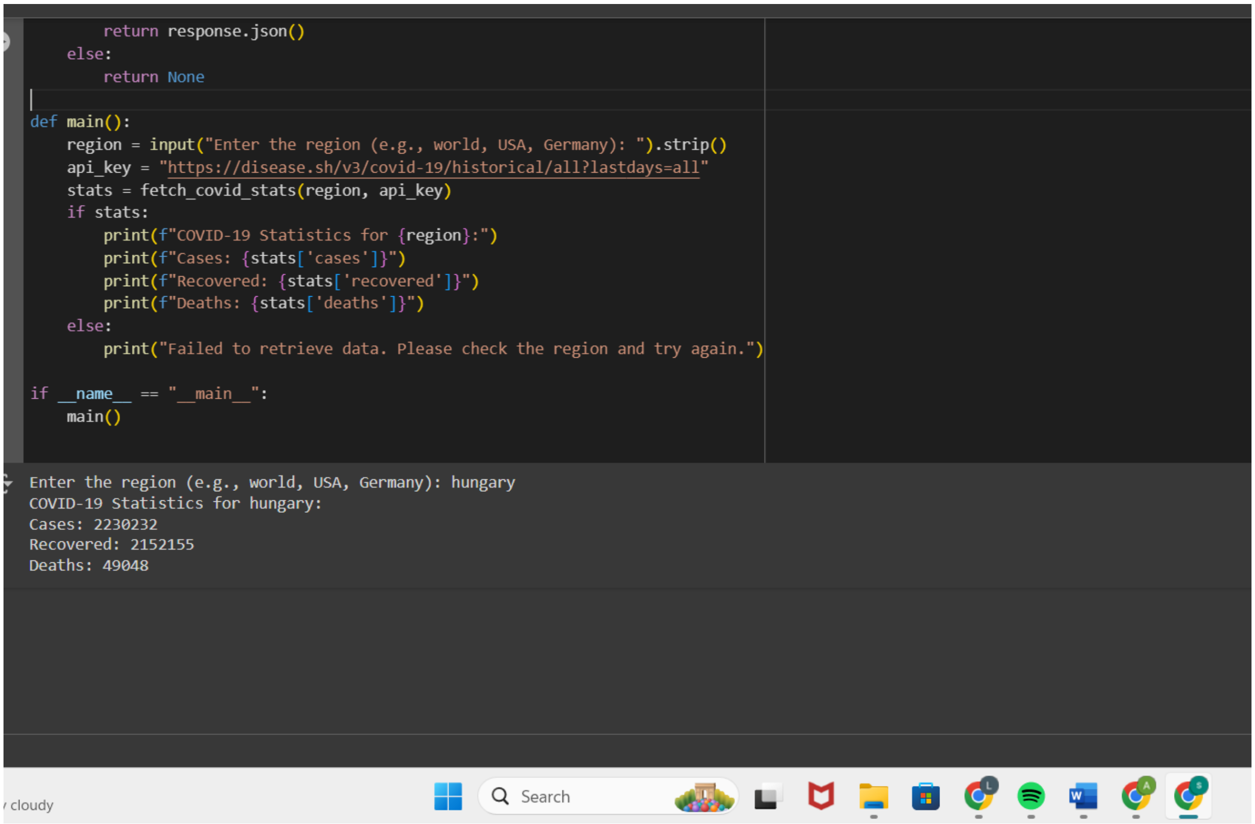


Implementation:

```
import requests


def fetch_covid_stats(region, api_key):
    base_url = "https://disease.sh/v3/covid-19"
    headers = {"Authorization": f"Bearer {api_key}"}
    response = requests.get(f"{base_url}/all" if region == "world" else f"{base_url}/countries/{region}", headers=headers)
    if response.status_code == 200:
        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()
```

Output:

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



```
        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
```