KARTHEEKA.REPALLE

192372289

CSE-AI

PYTHON API PROGRAMS DOCUMENTATION

DATE : 16/07/2024

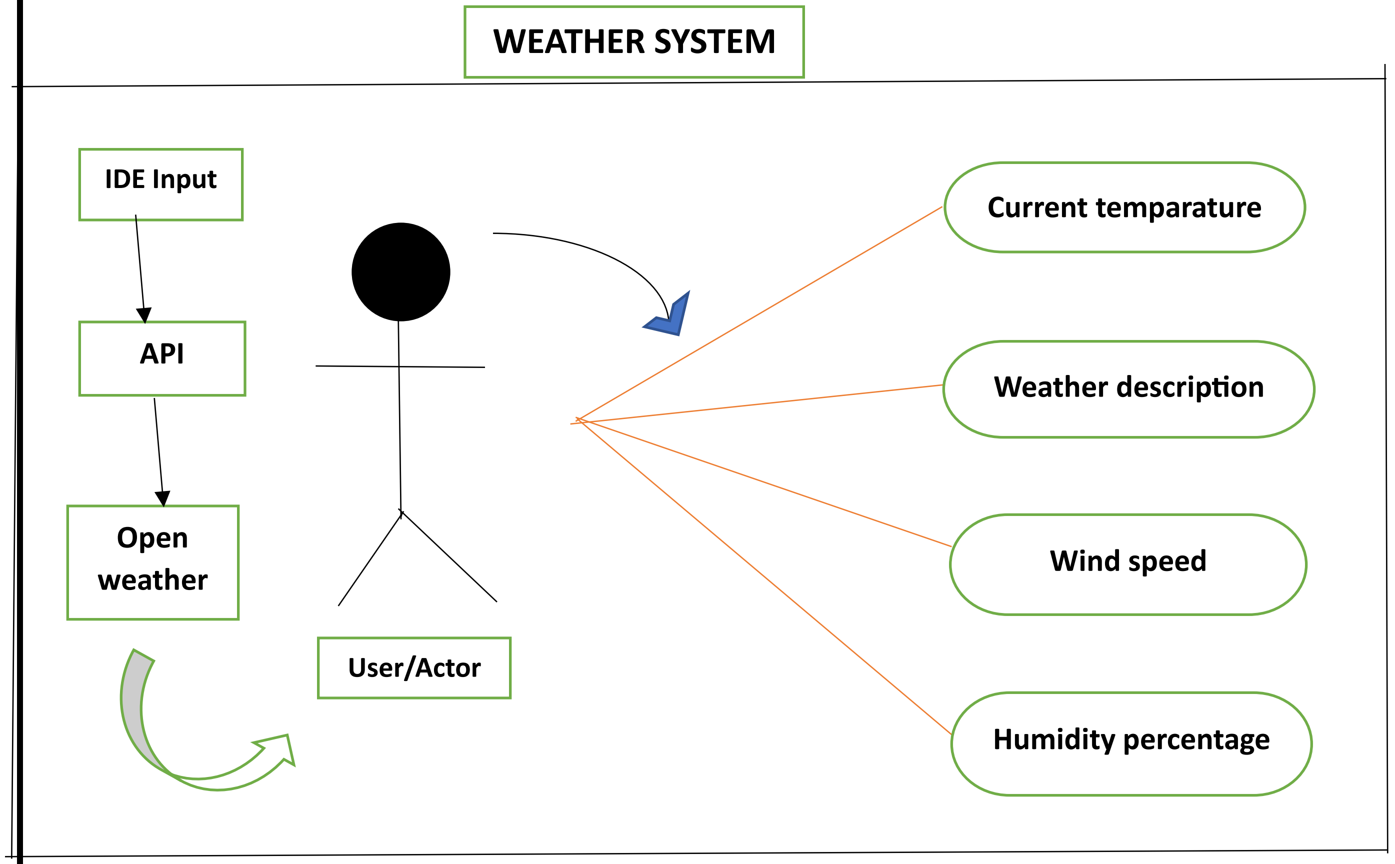
1. Real-Time Weather Monitoring System

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.

Data flow diagram:



Implementation

```
import requests

def fetch_weather(location):
    api_key = '59536a9c20d5a3c9fba920b884b7d2f3'  # Replace with your OpenWeatherMap
API key
    base_url = 'http://api.openweathermap.org/data/2.5/weather'
    params = {
        'q': location,
        'appid': api_key,
        'units': 'metric'
    }
    try:
        response = requests.get(base_url, params=params)
        if response.status_code == 200:
            data = response.json()
            weather_description = data['weather'][0]['description']
            temperature = data['main']['temp']
            humidity = data['main']['humidity']
            wind_speed = data['wind']['speed']
            print(f'Weather in {location}:')
            print(f'- Description: {weather_description}')
            print(f'- Temperature: {temperature} °C')
            print(f'- Humidity: {humidity}%')
            print(f'- Wind Speed: {wind_speed} m/s')
        else:
            print(f'Error fetching data: {response.status_code}')
    except requests.exceptions.RequestException as e:
        print(f'Error fetching data: {e}')
def main():
    location = input("Enter city name or coordinates (latitude,longitude): ")
    fetch_weather(location)
if __name__ == "__main__":
    main()
```

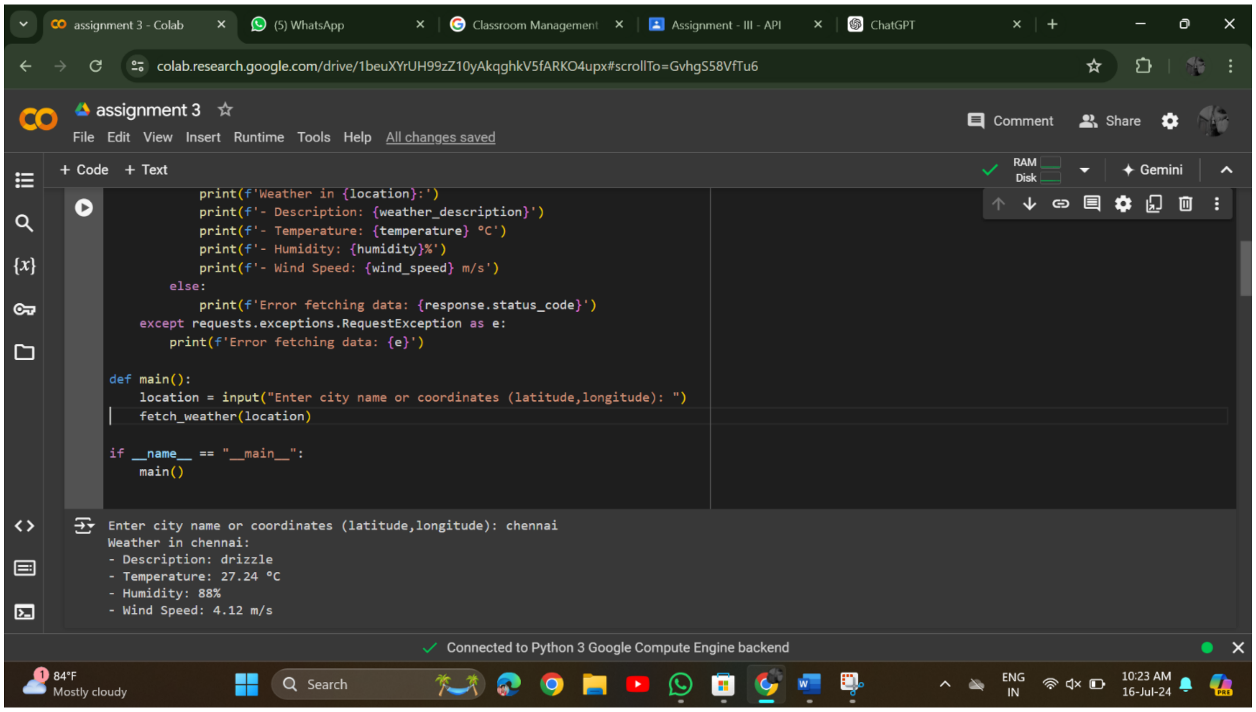
Displaying data:

Input:

```
Enter city name or coordinates(latitude,longitude):
 Chennai
```

Output:

```
Weather in chennai:
      Description: drizzle
      Temperature: 27.24 °C
      Humidity: 88%
      Wind Speed: 4.12 m/s
```



```
            print(f'Weather in {location}:')
            print(f'- Description: {weather_description}')
            print(f'- Temperature: {temperature} °C')
            print(f'- Humidity: {humidity}%')
            print(f'- Wind Speed: {wind_speed} m/s')
        else:
            print(f'Error fetching data: {response.status_code}')
    except requests.exceptions.RequestException as e:
        print(f'Error fetching data: {e}')

def main():
    location = input("Enter city name or coordinates (latitude,longitude): ")
    fetch_weather(location)

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

```
Enter city name or coordinates (latitude,longitude): chennai
Weather in chennai:
- Description: drizzle
- Temperature: 27.24 °C
- Humidity: 88%
- Wind Speed: 4.12 m/s
```

2. Inventory Management System Optimization

Scenario:

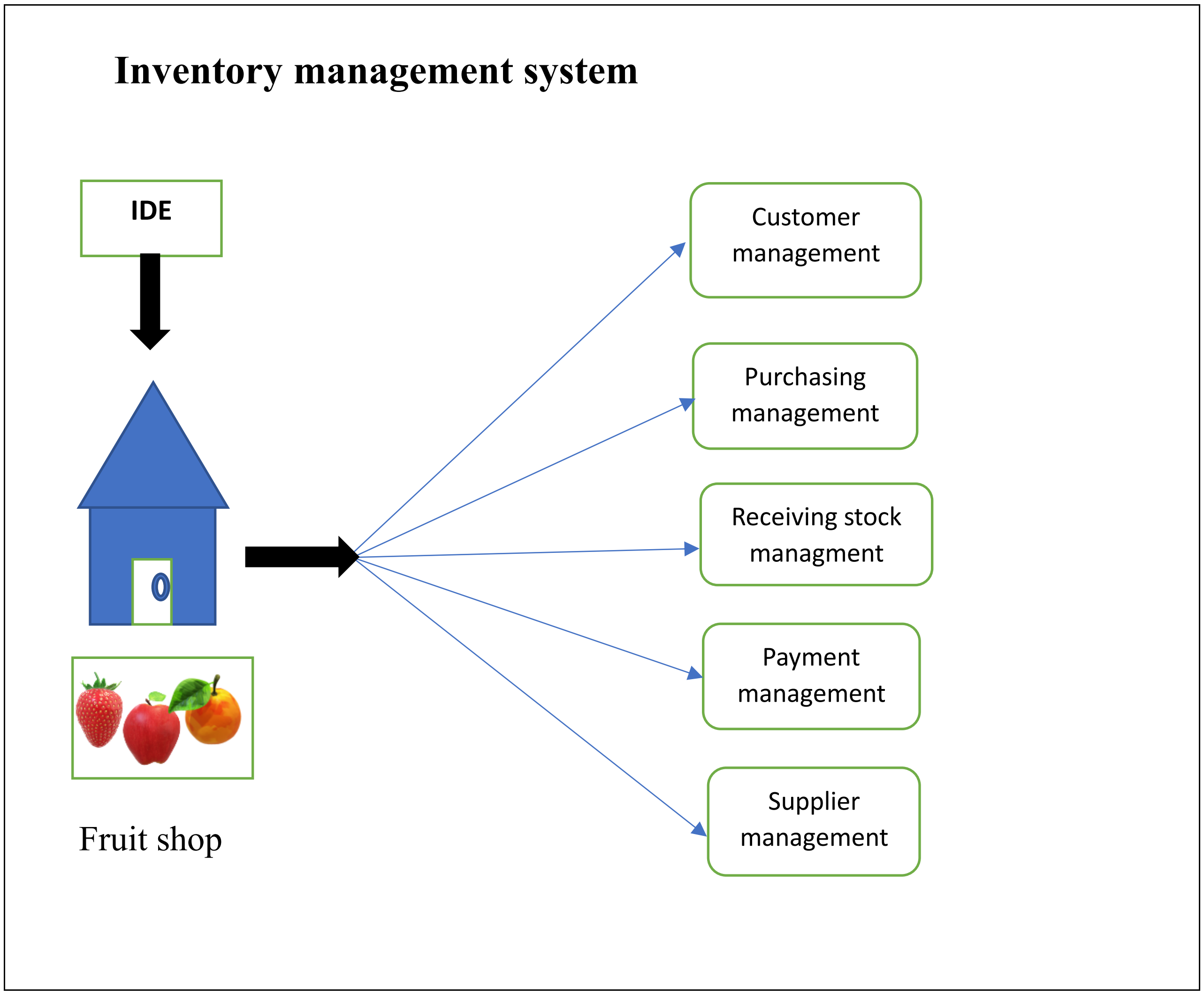
You have been hired by a retail company to optimize their inventory management system. The company wants to minimize stockouts and overstock situations while maximizing inventory
turnover and profitability.

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:

```
class InventoryManager:
    def __init__(self):
        self.inventory = {}

    def add_item(self, item_name, quantity, price):
        if item_name in self.inventory:
            # Item already exists, update quantity and price
            self.inventory[item_name]['quantity'] += quantity
            self.inventory[item_name]['price'] = price
        else:
            # Add new item to inventory
            self.inventory[item_name] = {'quantity': quantity, 'price': price}

    def remove_item(self, item_name):
        if item_name in self.inventory:
            del self.inventory[item_name]
        else:
            print(f"{item_name} not found in inventory.")

    def update_item_quantity(self, item_name, new_quantity):
        if item_name in self.inventory:
            self.inventory[item_name]['quantity'] = new_quantity
        else:
            print(f"{item_name} not found in inventory.")

    def get_inventory_value(self):
        total_value = 0
        for item_name, details in self.inventory.items():
            total_value += details['quantity'] * details['price']
        return total_value

    def print_inventory(self):
        print("Inventory:")
        for item_name, details in self.inventory.items():
            print(f"{item_name}: Quantity - {details['quantity']}, Price - {details['price']}")

# Example usage:
manager = InventoryManager()
manager.add_item("strawberry", 500, 2.5)
manager.add_item("bluberry", 200, 0.5)
manager.print_inventory()

manager.update_item_quantity("strawberry", 150)
manager.print_inventory()

manager.remove_item("Blueberry")
manager.print_inventory()

print("Total Inventory Value:", manager.get_inventory_value())
```

Displaying data:

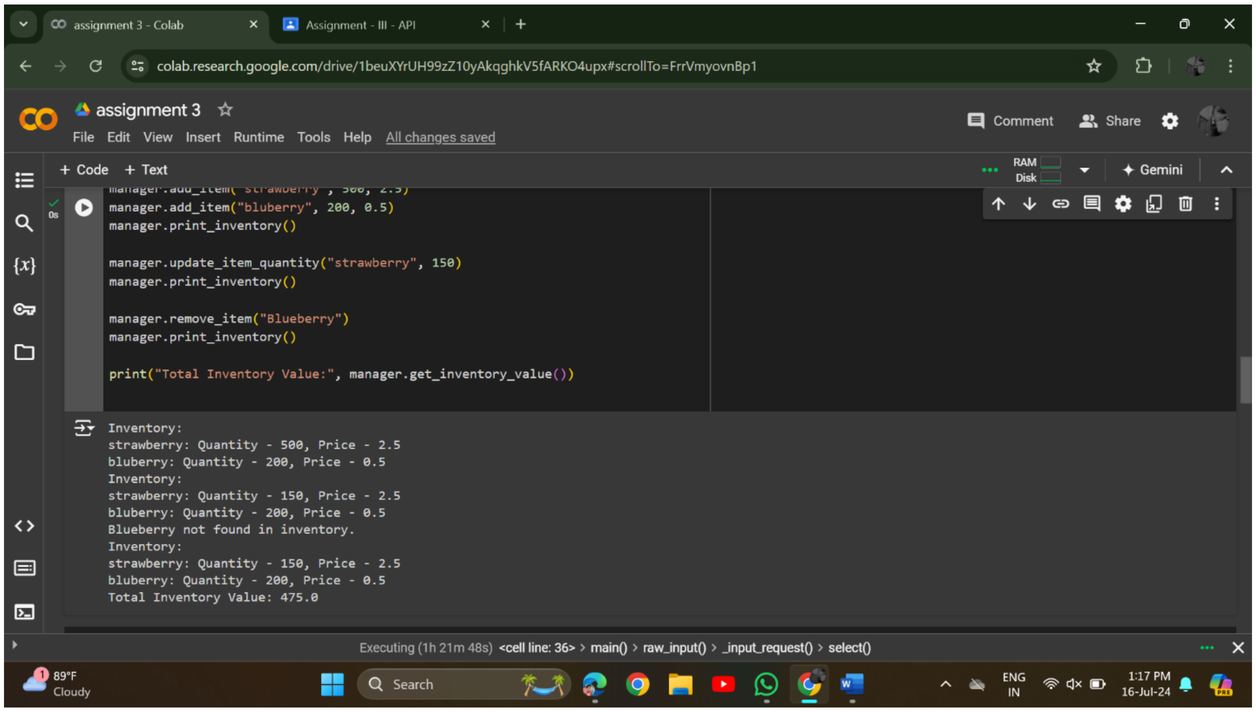
Input:

Item name: Strawberry, Quantity:500, Price: 2.5

Item name: Blueberry, Quantity:200, Price:0.5

Output:

```
Inventory:
  strawberry: Quantity - 500, Price - 2.5
  bluberry: Quantity - 200, Price - 0.5
Inventory:
  strawberry: Quantity - 150, Price - 2.5
  bluberry: Quantity - 200, Price - 0.5
Blueberry not found in inventory.
Inventory:
  strawberry: Quantity - 150, Price - 2.5
  bluberry: Quantity - 200, Price - 0.5
Total Inventory Value: 475.0
```


assignment 3 - Colab
Assignment - III - API
colab.research.google.com/drive/1beuXYrUH99zZ10yAkqghkV5fARKO4upx#scrollTo=FrrVmyovnBp1
assignment 3
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
RAM Disk
Gemini

```
manager.add_item("bluberry", 200, 0.5)
manager.print_inventory()

manager.update_item_quantity("strawberry", 150)
manager.print_inventory()

manager.remove_item("Blueberry")
manager.print_inventory()

print("Total Inventory Value:", manager.get_inventory_value())
```

```
Inventory:
strawberry: Quantity - 500, Price - 2.5
bluberry: Quantity - 200, Price - 0.5
Inventory:
strawberry: Quantity - 150, Price - 2.5
bluberry: Quantity - 200, Price - 0.5
Blueberry not found in inventory.
Inventory:
strawberry: Quantity - 150, Price - 2.5
bluberry: Quantity - 200, Price - 0.5
Total Inventory Value: 475.0
```

Executing (1h 21m 48s) <cell line: 36> > main() > raw_input() > _input_request() > select()
89°F Cloudy
Search
ENG IN
1:17 PM 16-Jul-24

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.

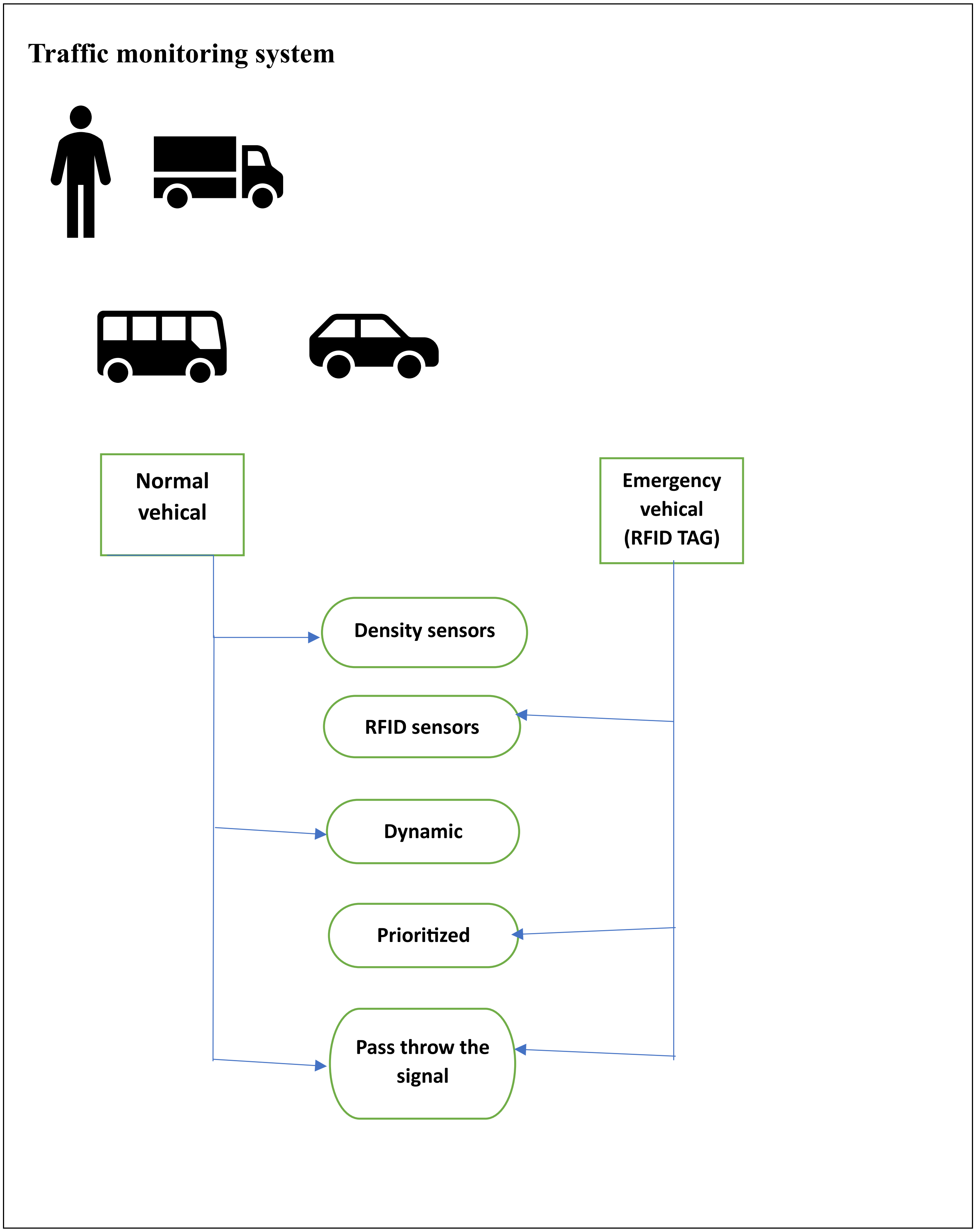
Tasks:

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 data
- Explanation of any assumptions made and potential improvements.

Data flow analysis:



Implementation:

```
import requests

def fetch_traffic_data(api_key, origin, destination):
    url = f"https://maps.googleapis.com/maps/api/directions/json?origin={origin}&destination=
{destination}&key={api_key}&departure_time=now&traffic_model=best_guess&alternatives=true"

    try:
        response = requests.get(url)
        data = response.json()
        return data
    except requests.exceptions.RequestException as e:
        print(f"Error fetching traffic data: {e}")
        return None

def display_traffic_data(traffic_data):
    if traffic_data is None or 'routes' not in traffic_data:
        print("No traffic data available.")
        return

    routes = traffic_data['routes']
    print(f"Total routes found: {len(routes)}\n")

    for idx, route in enumerate(routes, start=1):
        print(f"Route {idx}:")
        print(f"   - Distance: {route['legs'][0]['distance']['text']}")
        print(f"   - Duration in current traffic: {route['legs'][0]['duration_in_traffic']['text']}")
        print(f"   - Summary: {route['summary']}\n")

def main():
    api_key = 'YOUR_API_KEY'
    origin = input("Enter starting point (e.g., 'New York, NY'): ")
    destination = input("Enter destination (e.g., 'Los Angeles, CA'): ")

    traffic_data = fetch_traffic_data(api_key, origin, destination)
    display_traffic_data(traffic_data)

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

Display data:

Input:

```
Enter starting point (e.g., 'New York, NY'): andhra pradesh

Enter destination (e.g., 'Los Angeles, CA'): chennai
```

Output:

```
Total routes found= 20
```

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:

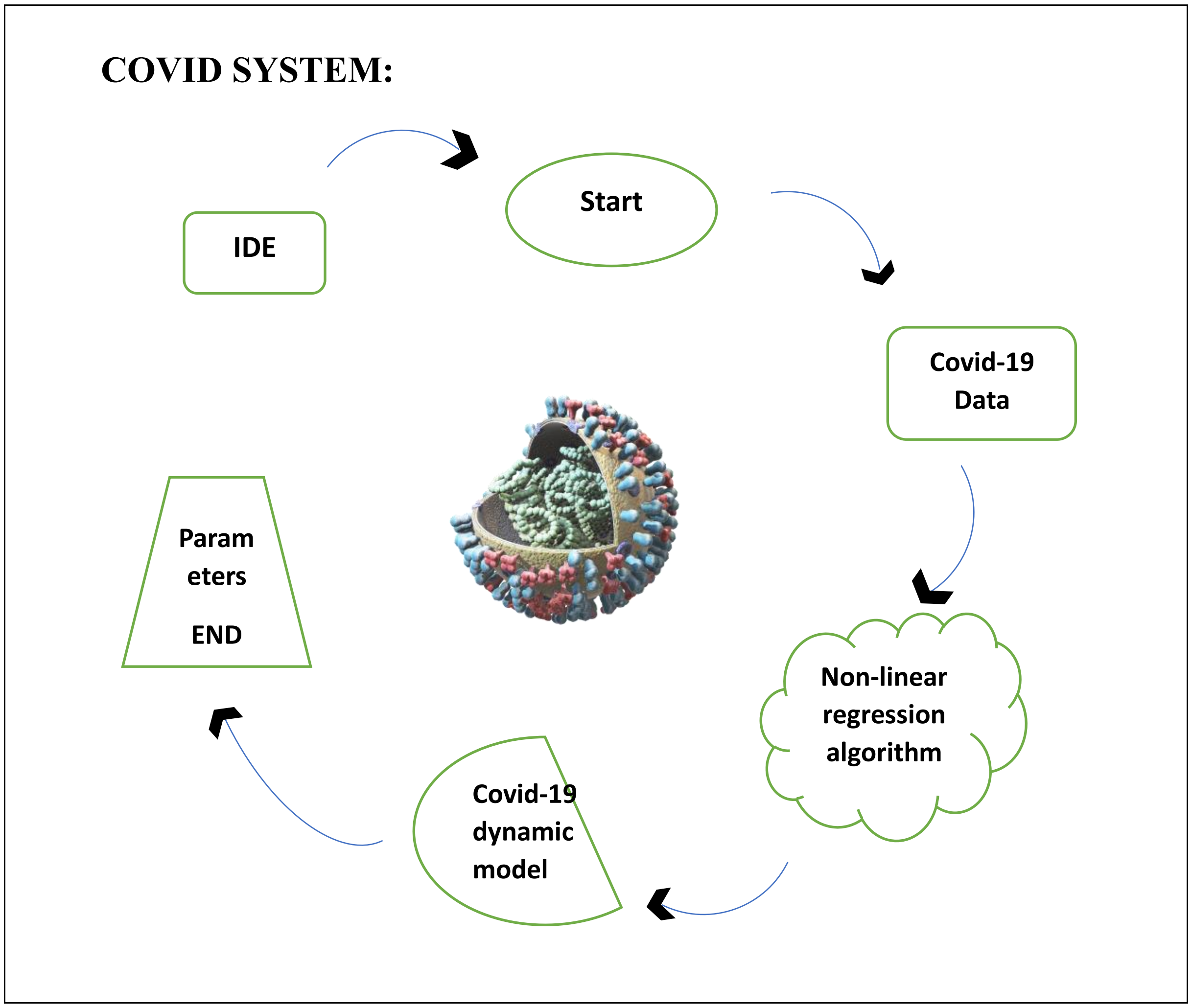
• Data flow diagram illustrating the interaction between the application and the API.

• 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.

Data flow diagram:



Implementation:

```
import requests

def fetch_covid_stats(region):
    url = f"https://disease.sh/v3/covid-19/countries/{region}"

      try:
          response = requests.get(url)
          data = response.json()
                return data
           except requests.exceptions.RequestException as e:
                print(f"Error fetching COVID-19 data: {e}")
                   return None
            def display_covid_stats(data):
                if data is None:
                    print("No data available.")
                    return

                country = data.get('country')
                cases = data.get('cases')
                recovered = data.get('recovered')
                deaths = data.get('deaths')

                print(f"COVID-19 Statistics for {country}:")
                print(f" - Total Cases: {cases}")
                print(f" - Recovered: {recovered}")
                print(f" - Deaths: {deaths}")

            def main():
                region = input("Enter a country name or 'all' for global statistics: ").lower()

                if region == 'all':
                    region = 'all'
                else:
                    # URL encoding for spaces in country names
                    region = region.replace(" ", "%20")

                covid_data = fetch_covid_stats(region)
                display_covid_stats(covid_data)

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

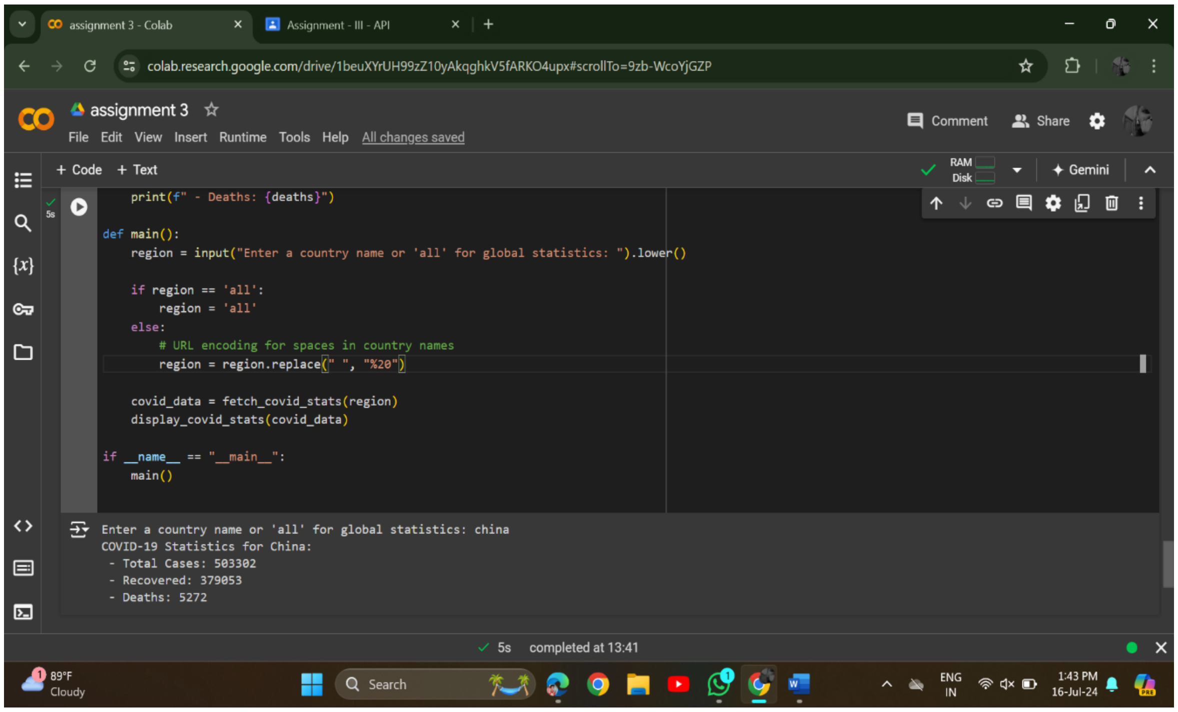
Display data:

Input:

Enter a country name or 'all' for global statistics: china

Output:

COVID-19 Statistics for China:
- Total Cases: 503302
- Recovered: 379053
- Deaths: 5272


assignment 3 - Colab
Assignment - III - API
colab.research.google.com/drive/1beuXYrUH99zZ10yAkqghkV5fARKO4upx#scrollTo=9zb-WcoYjGZP
assignment 3
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
RAM Disk
Gemini

```
    print(f" - Deaths: {deaths}")

def main():
    region = input("Enter a country name or 'all' for global statistics: ").lower()

    if region == 'all':
        region = 'all'
    else:
        # URL encoding for spaces in country names
        region = region.replace(" ", "%20")

    covid_data = fetch_covid_stats(region)
    display_covid_stats(covid_data)

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

```
Enter a country name or 'all' for global statistics: china
COVID-19 Statistics for China:
 - Total Cases: 503302
 - Recovered: 379053
 - Deaths: 5272
```