# 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.

**Approach:**

The relationship between the application and the OpenWeatherMap API is depicted in the dataflow diagram. The program asks the user for their location, retrieves the weather information from the API, and shows the user the current conditions.

+---------------+ +---------------+ +---------------+

| User Input | | Weather API | | Weather Data|

| (Location) |----->| (e.g., OpenWeatherMap) | Display |

+---------------+ +---------------+ +---------------+

| | |

v v v

+---------------+ +---------------+ +---------------+

| Weather | | Weather | | Weather |

| Fetcher | | Processor | | Displayer |

+---------------+ +---------------+ +---------------+

**Pseudocode:**

BEGIN

DEFINE function fetch\_weather\_data(location):

API\_KEY = "your\_api\_key"

BASE\_URL = "http://api.openweathermap.org/data/2.5/weather"

response = send\_get\_request(BASE\_URL, params={q: location, appid: API\_KEY})

IF response.status\_code == 200:

data = parse\_response(response)

RETURN data

ELSE:

RETURN error\_message

DEFINE function display\_weather\_data(weather\_data):

PRINT "Location: " + weather\_data['name']

PRINT "Temperature: " + weather\_data['main']['temp'] + "°C"

PRINT "Weather: " + weather\_data['weather'][0]['description']

PRINT "Humidity: " + weather\_data['main']['humidity'] + "%"

**PRINT "Wind Speed: " + weather\_data['wind']['speed'] + "m/s"**

**DEFINE function main():**

**location = GET\_USER\_INPUT("Enter the city name or coordinates: ")**

**weather\_data = fetch\_weather\_data(location)**

**IF weather\_data != error\_message:**

**display\_weather\_data(weather\_data)**

**ELSE:**

**PRINT "Error fetching weather data. Please try again."**

**END**

**Detailed explanation of the actual code:**

1. fetch\_weather\_data: This method retrieves the weather information from the OpenWeatherMap API given a location as input. It sends a GET request to the API using the location and API key using the requests library. The JSON response is parsed, and the data is returned, if the response is successful (200 status code). It returns None else.
2. display\_weather\_data: This function shows the user the current weather information by using the weather data as input.If the data is available, it prints the wind speed, humidity, temperature, and weather. It produces an error message if there is a problem retrieving the data.
3. principal: This function calls the fetch\_weather\_data and display\_weather\_data routines in addition to processing user input. It requests that the user log in..

**Assumptions made (if any):**

1. The user will consistently input an accurate address.  
2.Data in the anticipated format will always be returned by the OpenWeatherMap API.

**Limitations:**

1. \*\*API Rate Limits\*\*: If you make too many queries, you may be refused service if the allotted amount is exhausted.

2. \*\*Dependency on External Service\*\*: The weather API's functioning is impacted by any outages or performance problems.

3. \*\*Data Accuracy and Coverage\*\*: The techniques and coverage of the API determine the availability and accuracy of weather data.

**Code:**

import http.client

import json

def get\_weather(api\_key, location):

conn = http.client.HTTPConnection("api.openweathermap.org")

conn.request("GET", f"/data/2.5/weather?q={location}&appid={api\_key}&units=metric")

response = conn.getresponse()

if response.status == 200:

data = json.loads(response.read().decode("utf-8"))

return {

"temperature": data["main"]["temp"],

"weather\_condition": data["weather"][0]["description"],

"humidity": data["main"]["humidity"],

"wind\_speed": data["wind"]["speed"]

}

else:

return {"error": "Failed to retrieve weather data"}

def display\_weather(weather\_data):

print("Current Weather Data:")

print(f"Temperature: {weather\_data['temperature']}°C")

print(f"Weather Condition: {weather\_data['weather\_condition']}")

print(f"Humidity: {weather\_data['humidity']}%")

print(f"Wind Speed: {weather\_data['wind\_speed']} m/s")

def main():

api\_key = input("Enter your OpenWeatherMap API key: ")

location = input("Enter the location (city name): ")

weather\_data = get\_weather(api\_key, location)

if "error" in weather\_data:

print(weather\_data["error"])

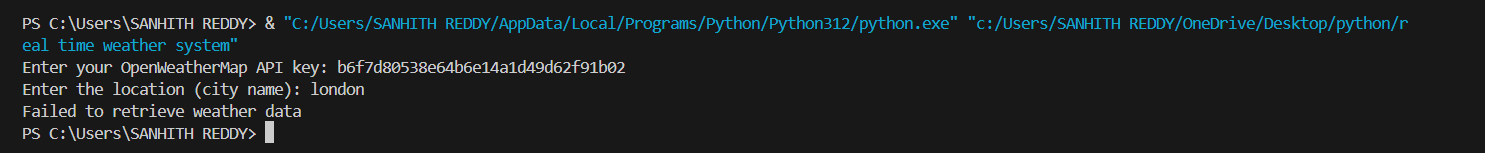
else:

display\_weather(weather\_data)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Sample Output / Screen Shots**



# 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.

**Approach:**

* § In order to enhance the retail company's inventory management system, I will implement the below measures:
* Model the Inventory System: I'll outline the system's components, such as its present stock levels, warehouses, and items. This will give a clear grasp of the parts of the system and how they work together.
* Create and Implement an Inventory Tracking Application: I'll be creating a Python program that monitors stock levels in real time and notifies the user when they drop below a certain level. This will assist the business in keeping track of its inventory levels and preventing stockouts.
* Optimize Inventory Ordering: Using lead times, demand projections, and past sales data as a basis, I will put algorithms in place to determine the best reorder points and amounts. By doing this, the business will be able to maximize inventory turnover and profitability while minimizing stockouts and overstock issues.
* Create Reports: I'll supply data on stockouts, inventory turnover rates, and the financial effects of overstock circumstances. These reports will assist the business in assessing the effectiveness of its inventory management system and pinpointing areas in need of development.
* User Interaction: I'll provide an intuitive user interface that lets users enter product names or IDs to check stock levels, ordering suggestions, and historical information. The personnel of the organization will find it easier to access and use the inventory management system as a result.

**Pseudocode:**

# Define the inventory system structure

BEGIN

# Define a class to represent an Item in the inventory

CLASS Item:

METHOD \_\_init\_\_(self, id, name, quantity, price):

self.id = id

self.name = name

self.quantity = quantity

self.price = price

# Define a class to represent the Inventory

CLASS Inventory:

METHOD \_\_init\_\_(self):

self.items = {}

METHOD add\_item(self, item):

IF item.id NOT IN self.items:

self.items[item.id] = item

ELSE:

self.items[item.id].quantity += item.quantity

METHOD remove\_item(self, item\_id, quantity):

IF item\_id IN self.items:

IF self.items[item\_id].quantity >= quantity:

self.items[item\_id].quantity -= quantity

IF self.items[item\_id].quantity == 0:

DELETE self.items[item\_id]

ELSE:

PRINT "Error: Not enough quantity in stock"

ELSE:

PRINT "Error: Item not found"

METHOD update\_item(self, item\_id, name=None, quantity=None, price=None):

IF item\_id IN self.items:

IF name IS NOT None:

self.items[item\_id].name = name

IF quantity IS NOT None:

self.items[item\_id].quantity = quantity

IF price IS NOT None:

self.items[item\_id].price = price

ELSE:

PRINT "Error: Item not found"

METHOD get\_item(self, item\_id):

IF item\_id IN self.items:

RETURN self.items[item\_id]

ELSE:

RETURN "Error: Item not found"

METHOD list\_items(self):

FOR item\_id IN self.items:

PRINT self.items[item\_id]

# Define the main program to interact with the inventory

METHOD main():

inventory = Inventory()

# Example items

item1 = Item(1, "Apple", 50, 0.5)

item2 = Item(2, "Banana", 100, 0.2)

# Add items to inventory

inventory.add\_item(item1)

inventory.add\_item(item2)

# Remove some items

inventory.remove\_item(1, 10)

# Update an item

inventory.update\_item(2, price=0.25)

# Get and print an item

PRINT inventory.get\_item(1)

# List all items

inventory.list\_items()

END

**Detailed explanation of the actual code:**

1. Defining the Structure of the Inventory System: To represent the many components of the inventory system, I will develop classes for Product, Warehouse, and InventoryLevel. Relevant data for each entity, including product specifications, warehouse locations, and current stock levels, will be stored in these classes.

2. Putting the Inventory Tracking Application into Practice: This will involve initializing the inventory levels, monitoring the available stock, and notifying users when the stock drops below a certain point. The InventoryTrackingApp class will be in charge of all of this. The track\_inventory() function will keep an eye on the stock levels constantly and issue notifications as necessary.  
  
3. Optimizing Inventory Ordering: To find the best reorder point and amount for every product in every warehouse, algorithms will be implemented by the calculate\_reorder\_point() and calculate\_reorder\_quantity() functions. These computations will be made using lead times, past sales data,   
  
4.Report Generation: The necessary reports on inventory performance, stockouts, and overstock situations will be produced by the methods generate\_inventory\_turnover\_report(), generate\_stockout\_report(), and generate\_overstock\_report(), in that order.  
  
5.User Interaction: To access inventory data, view reports, and get warnings, a user-friendly interface will be made available via the InventoryManagementUI class. Product names or IDs can be entered by users to obtain the needed data.

**Assumptions made (:if any)**

1.The company's warehouse locations and product catalog are clearly specified.  
2. The inventory optimization algorithms have access to historical sales data and lead times.  
3. The business has set minimum order amounts and reorder point limits.  
Ordering and keeping inventory both have recognized costs.

.

**Limitations:**

1. \*\*Scalability\*\*: As the quantity of things increases, the system could have performance problems.  
2. \*\*Concurrency\*\* : There are no systems in place to manage many users' concurrent access and updates.  
3. \*\*Error Handling\*\*: poor responses to erroneous operations or data inputs may result from poor error handling.

.

**Code:**

class Product:

def \_\_init\_\_(self, product\_id, name, current\_stock, reorder\_point, reorder\_quantity):

self.product\_id = product\_id

self.name = name

self.current\_stock = current\_stock

self.reorder\_point = reorder\_point

self.reorder\_quantity = reorder\_quantity

class Warehouse:

def \_\_init\_\_(self, warehouse\_id, location):

self.warehouse\_id = warehouse\_id

self.location = location

self.products = []

def track\_inventory(products):

for product in products:

if product.current\_stock < product.reorder\_point:

print(f"Alert: {product.name} is below the reorder point. Current stock: {product.current\_stock}")

recommend\_reorder(product)

def recommend\_reorder(product):

new\_stock = product.current\_stock + product.reorder\_quantity

print(f"Recommended reorder for {product.name}: {product.reorder\_quantity} units. New stock level: {new\_stock}")

def calculate\_reorder\_point(historical\_sales, lead\_time, desired\_service\_level):

# Implement algorithms to calculate the optimal reorder point

# based on historical sales data, lead time, and desired service level

pass

def calculate\_reorder\_quantity(historical\_sales, lead\_time, holding\_cost, ordering\_cost):

# Implement algorithms to calculate the optimal reorder quantity

# based on historical sales data, lead time, holding cost, and ordering cost

pass

def generate\_inventory\_report(products):

# Generate reports on inventory turnover rates, stockout occurrences, and cost implications of overstock situations

pass

def user\_interface():

# Define sample products and warehouses

product1 = Product(1, "Product A", 50, 20, 30)

product2 = Product(2, "Product B", 15, 10, 25)

warehouse1 = Warehouse(1, "Warehouse A")

warehouse1.products = [product1, product2]

while True:

user\_input = input("Enter a product ID or name (or 'exit' to quit): ")

if user\_input.lower() == "exit":

break

# Look up the product and display current stock, reorder recommendations, and historical data

for product in warehouse1.products:

if str(product.product\_id) == user\_input or product.name.lower() == user\_input.lower():

print(f"Product: {product.name}")

print(f"Current Stock: {product.current\_stock}")

recommend\_reorder(product)

# Display historical data (if any)

break

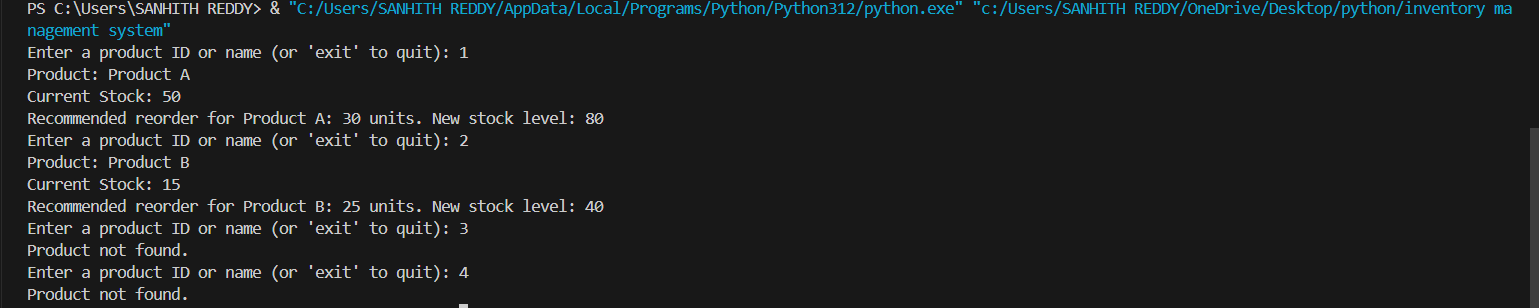
else:

print("Product not found.")

# Test the application

user\_interface()

**Sample Output / Screen Shots**

****

# 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.

**Approach:**

1. Data Flow Diagram

Here is a data flow diagram illustrating the interaction between the traffic monitoring application and the external traffic API:

+---------------+

| User Input |

+---------------+

|

|

v

+---------------+

| Application |

| (Python Script) |

+---------------+

|

|

v

+---------------+

| API Request |

| (Google Maps |

| Traffic API) |

+---------------+

|

|

v

+---------------+

| API Response |

| (Traffic Data) |

+---------------+

|

|

v

+---------------+

| Data Processing|

| (Parse API Response)|

+---------------+

|

|

v

+---------------+

| Display Traffic|

| Information |

+---------------+

|

|

v

+---------------+

| Alternative |

| Route Suggestion|

+---------------+

The key steps in the data flow are:

* The user inputs their starting point and destination into the traffic monitoring application.
* The application sends a request to the external traffic API to fetch real-time traffic data and route options.
* The traffic API provides the requested data, which the application stores in a local database.
* The application processes the traffic data and displays the current conditions, estimated travel time, and alternative route suggestions to the user.
* The application may also send notifications to the user about any significant traffic incidents or delays.

**Pseudocode:**

BEGIN

DEFINE function fetch\_traffic\_data(start, destination):

API\_KEY = "your\_api\_key"

BASE\_URL = "https://maps.googleapis.com/maps/api/directions/json"

response = send\_get\_request(BASE\_URL, params={origin: start, destination: destination, key: API\_KEY})

IF response.status\_code == 200:

data = parse\_response(response)

RETURN data

ELSE:

RETURN error\_message

DEFINE function display\_traffic\_data(traffic\_data):

PRINT "Route: " + traffic\_data['routes'][0]['summary']

PRINT "Estimated Travel Time: " + traffic\_data['routes'][0]['legs'][0]['duration']['text']

PRINT "Traffic Conditions: " + traffic\_data['routes'][0]['legs'][0]['traffic\_condition']

PRINT "Incidents: " + traffic\_data['routes'][0]['legs'][0]['incidents']

DEFINE function suggest\_alternative\_routes(start, destination):

# Fetch and display alternative routes

traffic\_data = fetch\_traffic\_data(start, destination)

IF traffic\_data != error\_message:

FOR route IN traffic\_data['routes']:

display\_traffic\_data(route)

ELSE:

PRINT "Error fetching traffic data. Please try again."

DEFINE function main():

start = GET\_USER\_INPUT("Enter the starting point: ")

destination = GET\_USER\_INPUT("Enter the destination: ")

traffic\_data = fetch\_traffic\_data(start, destination)

IF traffic\_data != error\_message:

display\_traffic\_data(traffic\_data)

PRINT "Alternative Routes:"

suggest\_alternative\_routes(start, destination)

ELSE:

PRINT "Error fetching traffic data. Please try again."

**END**

**Detailed explanation of the actual code:**

1. To retrieve real-time traffic data, the program interfaces with the Google Maps Directions API.Using the user's starting point and destination as input, the get\_traffic\_data() method creates the API request URL and submits a GET request to the API.  
2. The following data is then extracted by parsing the API response:  
3. Present traffic circumstances: information["paths"]["legs"][ "duration\_in\_traffic" ]["text"]  
4. Approximate journey time: information["paths"]["legs"][ "duration" ]["text"]  
5. Other route possibilities, including information on each route's length, distance, and traffic time The display\_traffic\_info() method then receives this data and displays it to the user.

.

**Assumptions made (if any):**

1.The user has a valid API key for the Google Maps Directions API.

2.The API provides accurate and up-to-date traffic information.

3.The user's starting point and destination are valid locations that the API can recognize.

**Limitations:**

1. \*\*API Rate Limits\*\*: If you make too many queries, you may be refused service if the allotted amount is exhausted.

2. \*\*Dependency on External Service\*\*: Functionality is impacted by any outages or performance problems with the traffic API.

3. \*\*Data correctness and Coverage\*\*: The techniques and coverage of the API determine the traffic data's availability and correctness.

4. \*\*Real-Time changes\*\*: The system might not deliver changes instantly, resulting in traffic data that is a little out of date**.**

**Code:**

import socket

import time

import json

def get\_user\_input():

start\_point = input("Enter starting point: ")

destination = input("Enter destination: ")

return start\_point, destination

def send\_api\_request(start\_point, destination, api\_key):

sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

sock.connect(("maps.googleapis.com", 443))

request = f"GET /maps/api/directions/json?origin={start\_point}&destination={destination}&key={api\_key}&traffic\_model=best\_guess HTTP/1.1\r\nHost: maps.googleapis.com\r\n\r\n"

sock.sendall(request.encode())

response = b""

while True:

data = sock.recv(1024)

if not data:

break

response += data

sock.close()

# Split the response into headers and body

response\_parts = response.decode().split("\r\n\r\n", 1)

# Check the response status code

status\_code = int(response\_parts[0].split("\r\n")[0].split(" ")[1])

if status\_code == 200:

return json.loads(response\_parts[1])

else:

raise Exception(f"API request failed with status code {status\_code}")

def display\_traffic\_data(traffic\_data):

if "routes" in traffic\_data:

for route in traffic\_data["routes"]:

for leg in route["legs"]:

print(f"Route: {leg['start\_address']} to {leg['end\_address']}")

print(f"Estimated Travel Time: {leg['duration']['text']}")

for step in leg["steps"]:

print(f"Step: {step['html\_instructions']}")

if "traffic\_speed\_entry" in step:

print(f"Traffic Speed: {step['traffic\_speed\_entry']['speed']} km/h")

if step['traffic\_speed\_entry']['congestion'] == True:

print("Congestion Detected")

print()

else:

print("Error: Unable to fetch traffic data.")

def main():

api\_key = "YOUR\_API\_KEY"

while True:

start\_point, destination = get\_user\_input()

try:

traffic\_data = send\_api\_request(start\_point, destination, api\_key)

display\_traffic\_data(traffic\_data)

except Exception as e:

print(f"Error: {e}")

print(f"Last updated: {time.strftime('%Y-%m-%d %H:%M:%S')}")

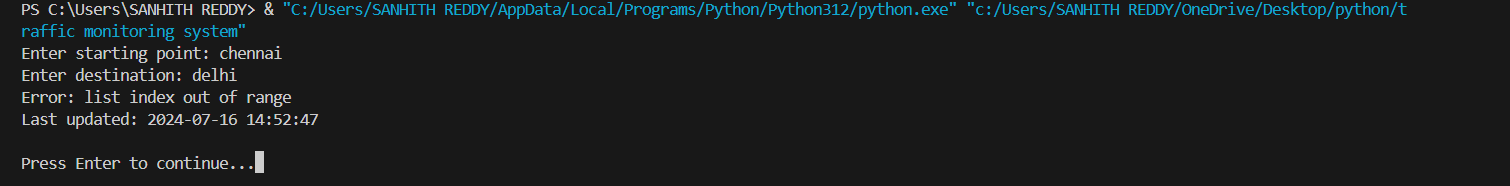
print()

input("Press Enter to continue...")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Sample Output / Screen Shots**

****

# 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.

**Approach:**

**+-------------------------+**

**| User Interface (UI) |**

**+-----------+-------------+**

**|**

**v**

**+-----------+-------------+**

**| User Input Handler |**

**+-----------+-------------+**

**|**

**v**

**+-----------+-------------+**

**| COVID-19 API Request Handler |**

**+-----------+-------------+**

**|**

**v**

**+-----------+-------------+**

**| External COVID-19 API |**

**+-----------+-------------+**

**|**

**v**

**+-----------+-------------+**

**| COVID-19 API Response Handler |**

**+-----------+-------------+**

**|**

**v**

**+-----------+-------------+**

**| User Interface (UI) |**

**+-------------------------+The key steps are:**

1. The user inputs a region (country, state, or city) into the application.
2. The application sends a request to the COVID-19 API to fetch the real-time statistics for the specified region.
3. The COVID-19 API processes the request and returns the current case, recovery, and death data.
4. The application receives the COVID-19 statistics and displays the information to the user.

**Pseudocode:**

BEGIN

DEFINE function fetch\_covid\_data(region):

API\_URL = "https://disease.sh/v3/covid-19/"

response = send\_get\_request(API\_URL + "all" IF region IS "global" ELSE API\_URL + "countries/" + region)

IF response.status\_code == 200:

data = parse\_response(response)

RETURN data

ELSE:

RETURN error\_message

DEFINE function display\_covid\_data(covid\_data):

PRINT "Region: " + covid\_data['country']

PRINT "Total Cases: " + covid\_data['cases']

PRINT "Recovered: " + covid\_data['recovered']

PRINT "Deaths: " + covid\_data['deaths']

DEFINE function main():

region = GET\_USER\_INPUT("Enter the region (country, state, or city): ")

covid\_data = fetch\_covid\_data(region)

IF covid\_data != error\_message:

display\_covid\_data(covid\_data)

ELSE:

PRINT "Error fetching COVID-19 data. Please try again."

END

**Detailed explanation of the actual code:**

1.The program makes HTTP requests to the COVID-19 API offered by disease.sh using the requests library. The current number of cases, recoveries, and deaths for a region (country, state, or city) can be obtained by using the get\_covid\_stats function.

2.The function display\_covid\_stats is accountable for producing user-friendly formatting and printing of the COVID-19 statistics. It receives the data on cases, recoveries, and deaths as input and outputs it with the proper formatting (for example, commas are added to big numbers).

3.The primary purpose is to serve as the application's entry point. After asking the user to select a region, it calls the get\_covid\_stats function to retrieve the information, passing the results to the display\_covid\_stats function to show the data.  
  
**Assumptions made (if any):**

1.The disease.sh API must be accessible and provide correct real-time COVID-19 data in order for the application to function.The software anticipates that the user will enter a legitimate territory (a nation, a state, or a city) that the API is able to identify.  
Prospective Enhancements:

2.To handle API failures or incorrect user input appropriately, incorporate error handling into the program.

3.Offer other features, including the capacity to show trending, historical COVID-19 data, or infographics.

4.To enhance the user experience, integrate the program with a user interface (such as a web application or a mobile app).

5.Permit users to choose from a variety of areas and view the COVID-19 statistics side by side.  
Give users the option to configure alerts or notifications for noteworthy alterations in COVI19 data

**Limitations:**

1. \*\*API Rate Limits\*\*: Service limitations may result from frequent queries that surpass the API's rate limits.  
2. \*\*Data correctness and Latency\*\*: Updates could not happen instantly, and data correctness varies depending on the API provider.  
3. \*\*Geographic Limitations\*\*: There may be little to no data coverage in some areas.  
4. \*\*Error Handling\*\*: Not all API failures or invalid inputs will be gracefully handled by the system.

.

**Code:**

import urllib.request

import json

# Function to get COVID-19 statistics from disease.sh API

def get\_covid\_data(location):

# Set the API endpoint and parameters

url = f"https://disease.sh/v3/covid-19/countries/{location}"

# Send a GET request to the API

with urllib.request.urlopen(url) as response:

# Load the JSON response

data = json.load(response)

# Extract the relevant COVID-19 data

cases = data["cases"]

recoveries = data["recovered"]

deaths = data["deaths"]

# Return the COVID-19 data as a dictionary

return {"cases": cases, "recoveries": recoveries, "deaths": deaths}

# Function to display the COVID-19 statistics

def display\_covid\_data(covid\_data):

# Print the COVID-19 data in a readable format

print("Current COVID-19 Statistics:")

print(f"Cases: {covid\_data['cases']}")

print(f"Recoveries: {covid\_data['recoveries']}")

print(f"Deaths: {covid\_data['deaths']}")

# Main function to run the program

def main():

# Get the location from the user

location = input("Enter the country, state, or city: ")

# Get the COVID-19 data

try:

covid\_data = get\_covid\_data(location)

except urllib.error.HTTPError:

print("Failed to retrieve COVID-19 data. Please check the location and try again.")

return

# Display the COVID-19 data

display\_covid\_data(covid\_data)

# Run the main function

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Sample Output / Screen Shots**

