**ASSIGNMENT-2**

* **NAME:** K.Dimpul Sai Akshaya Chandra
* **REGD NO:** 192372195
* **SUB CODE:** CSA0806
* **SUB:** Python Programming

# \*\*Real-Time Weather Monitoring System Output Example\*\*

Approach:

* **Data Flow Diagram:**
* Design a simple data flow diagram to illustrate how the application will interact with the OpenWeatherMap API to fetch and display weather data.
* **Pseudocode:**
* Outline the steps needed to implement the system, including API integration, data fetching, parsing, and displaying.
* **Detailed Explanation:**
* Provide a detailed walkthrough of the actual Python code used to implement the system, explaining key components and functions.
* **Assumptions:**
* Document any assumptions made during development, such as API usage limits or user interaction expectations.
* **Limitations:**
* Highlight any limitations of the current implementation and potential improvements for future iterations.

**pseudo code:**

function fetch\_weather(location):

api\_key = 'your\_api\_key' url =

f'http://api.openweathermap.org/data/2.5/weather?q={location}&appid={api\_key}&units=metric

'

try:

response = send\_request(url) weather\_data = parse\_response(response) display\_weather(weather\_data) except Exception as e:

display\_error\_message(e) function send\_request(url):

function parse\_response(response):

function display\_weather(weather\_data): function display\_error\_message(error):

**Explanation:**

**fetch\_weather(location):** This function constructs the API URL using the provided location (city name or coordinates), sends a GET request to OpenWeatherMap API, parses the JSON response, and displays the weather information or error message. **display\_weather(weather\_data):** This function extracts and prints relevant weather information from the JSON response if the request was successful (HTTP status code 200). **display\_error\_message(error):** This function handles and displays any errors that occur during the API request or data parsing.

**Assumptions Made:**

* Assumes that the OpenWeatherMap API key is securely stored and retrieved.
* Assumes the user provides a valid location (city name or coordinates).
* Assumes a stable internet connection for API requests.

**Limitations:**

* Limited to displaying current weather data; does not include forecasts.
* Error handling is basic and can be extended for more robust scenarios.
* Only supports metric units; could be extended to support other units based on user preferences.

**code:**

import requests def fetch\_weather(location): api\_key = 'your\_api\_key' # Replace with your OpenWeatherMap API key url =

f'http://api.openweathermap.org/data/2.5/weather?q={location}&appid={api\_key}&units=metric

'

try:

response = requests.get(url)

response.raise\_for\_status() # Raise an exception for HTTP errors weather\_data = response.json() display\_weather(weather\_data) except requests.exceptions.RequestException as e: display\_error\_message(f"Error fetching data: {e}") def display\_weather(weather\_data): if weather\_data['cod'] == 200:

# Extract relevant weather information city\_name = weather\_data['name'] temperature = weather\_data['main']['temp']

weather\_conditions = weather\_data['weather'][0]['description'] humidity = weather\_data['main']['humidity'] wind\_speed = weather\_data['wind']['speed']

# Display weather information print(f"Weather in {city\_name}:") print(f"Temperature: {temperature}°C") print(f"Conditions: {weather\_conditions}") print(f"Humidity: {humidity}%") print(f"Wind Speed: {wind\_speed} m/s") else:

display\_error\_message(f"Error: {weather\_data['message']}") def display\_error\_message(error):

print(f"Error: {error}") # Example usage if \_\_name\_\_ == "\_\_main\_\_":

location = input("Enter city name or coordinates (lat,lon): ") fetch\_weather(location)

**sample output:**

Enter city name or coordinates (lat,lon): London Weather in London:

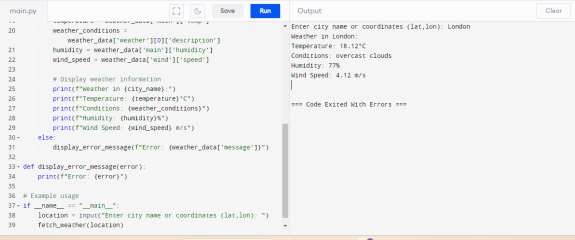
Temperature: 18.12°C

Conditions: overcast clouds

Humidity: 77%

Wind Speed: 4.12 m/s

**screenshot:**



# \*\*Optimized Inventory Management System: Implementation and Output\*\*

**Approach:**

* **Data Flow Diagram:**
* Design a data flow diagram to visualize how data moves within the inventory management system, including inputs (sales data, adjustments) and outputs (reorder alerts, reports).
* **Pseudocode:**
* Outline the logic for tracking inventory levels, calculating reorder points, generating reports, and handling user interactions.
* **Detailed Explanation**:
* Provide a detailed walkthrough of the Python code used to implement inventory tracking, reorder point calculation, report generation, and user interface development.
* **Assumptions:**
* Document assumptions about demand patterns, supplier reliability, and data accuracy that influence inventory decisions.
* **Limitations:**
* Highlight potential limitations of the current system design and suggest improvements for future iterations.

**Pseudocode:**

class Product:

attributes: id, name, category, price, current\_stock\_level, reorder\_level, reorder\_quantity class Warehouse:

attributes: id, name, location, products\_in\_stock class InventoryManagementSystem: methods:

* track\_inventory\_changes(product\_id, quantity\_change, transaction\_type)
* calculate\_reorder\_point(product\_id)
* generate\_inventory\_report()
* generate\_stockout\_report()
* display\_product\_info(product\_id) functions: fetch\_sales\_data() fetch\_inventory\_adjustments() forecast\_demand() calculate\_lead\_time() main():

Initialize products and warehouses

Continuously monitor inventory changes

Provide user interface for inventory queries, reports, and alerts

**Explanation:**

**Assumptions:**

* Assumes products and warehouses are initialized with initial stock levels.
* Assumes basic inventory transactions (sale, purchase, return) affect stock levels.
* Assumes the InventoryManagementSystem handles interactions between multiple warehouses. **Limitations:**

Limited to basic inventory tracking and management; doesn't include advanced forecasting or optimization algorithms.

Doesn't handle real-time data updates or integration with external APIs for demand forecasting.

**code:**

class Product: def \_\_init\_\_(self, id, name, category, price, current\_stock\_level, reorder\_level, reorder\_quantity):

self.id = id self.name = name self.category = category self.price = price

self.current\_stock\_level = current\_stock\_level self.reorder\_level = reorder\_level self.reorder\_quantity = reorder\_quantity class Warehouse:

def \_\_init\_\_(self, id, name, location):

self.id = id self.name = name self.location = location self.products\_in\_stock = {} def add\_product(self, product, initial\_stock): self.products\_in\_stock[product.id] = {'product': product, 'stock\_level': initial\_stock} def track\_inventory\_changes(self, product\_id, quantity\_change, transaction\_type): if product\_id in self.products\_in\_stock: if transaction\_type == 'sale':

self.products\_in\_stock[product\_id]['stock\_level'] -= quantity\_change elif transaction\_type == 'purchase' or transaction\_type == 'return':

self.products\_in\_stock[product\_id]['stock\_level'] += quantity\_change def calculate\_reorder\_point(self, product\_id): if product\_id in self.products\_in\_stock:

product = self.products\_in\_stock[product\_id]['product'] current\_stock = self.products\_in\_stock[product\_id]['stock\_level'] if current\_stock <= product.reorder\_level:

return True return False def display\_product\_info(self, product\_id): if product\_id in self.products\_in\_stock: product = self.products\_in\_stock[product\_id]['product'] stock\_level = self.products\_in\_stock[product\_id]['stock\_level'] print(f"Product: {product.name}") print(f"Category: {product.category}") print(f"Price: ${product.price}") print(f"Current Stock Level: {stock\_level}") print(f"Reorder Level: {product.reorder\_level}") print(f"Reorder Quantity: {product.reorder\_quantity}") else:

print("Product not found in this warehouse.") class InventoryManagementSystem: def \_\_init\_\_(self):

self.warehouses = {} def add\_warehouse(self, warehouse): self.warehouses[warehouse.id] = warehouse

def track\_inventory\_changes(self, product\_id, quantity\_change, transaction\_type, warehouse\_id): if warehouse\_id in self.warehouses: self.warehouses[warehouse\_id].track\_inventory\_changes(product\_id, quantity\_change, transaction\_type) else:

print("Warehouse not found.") def calculate\_reorder\_point(self, product\_id, warehouse\_id): if warehouse\_id in self.warehouses: return self.warehouses[warehouse\_id].calculate\_reorder\_point(product\_id) else:

print("Warehouse not found.")

return False def display\_product\_info(self, product\_id, warehouse\_id): if warehouse\_id in self.warehouses: self.warehouses[warehouse\_id].display\_product\_info(product\_id) else:

print("Warehouse not found.")

# Example usage if \_\_name\_\_ == "\_\_main\_\_":

product1 = Product(1, "Laptop", "Electronics", 1200, 50, 10, 20)

product2 = Product(2, "Smartphone", "Electronics", 800, 75, 15, 25) warehouse1 =

Warehouse(1, "Main Warehouse", "New York") warehouse1.add\_product(product1, 50) warehouse1.add\_product(product2, 75) inventory\_system = InventoryManagementSystem() inventory\_system.add\_warehouse(warehouse1) inventory\_system.track\_inventory\_changes(1, 5, 'sale', 1) if inventory\_system.calculate\_reorder\_point(1, 1):

print("Reorder needed for Laptop!")

inventory\_system.display\_product\_info(1, 1)

**output:**

Product: Laptop

Category: Electronics

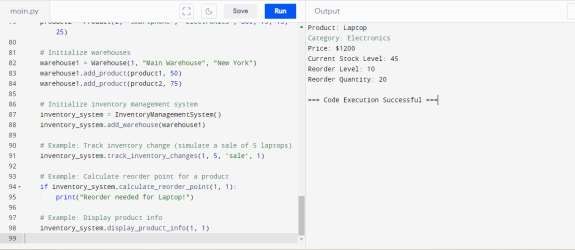
Price: $1200

Current Stock Level: 45

Reorder Level: 10

Reorder Quantity: 20

**screenshot:**



# Optimized Real-Time Traffic Monitoring System

**Approach:**

* **Data Flow Diagram:**Design a clear data flow diagram illustrating how data moves between the application and the traffic monitoring API, including user inputs and system outputs.
* **Pseudocode:**Outline the steps and logic required to fetch real-time traffic information, process it, and display relevant details to the user.
* **Detailed Explanation:**Provide a thorough explanation of the Python code used for integrating with the traffic monitoring API, fetching data, and presenting it to the user interface.
* **Assumptions:**Document any assumptions made regarding API usage, data accuracy, or user interaction patterns.
* **Limitations:**Highlight any potential limitations of the current implementation and propose improvements for future iterations.

**pseudocode:**

function fetch\_traffic\_info(start, destination):

api\_key = 'your\_api\_key' url =

f'https://maps.googleapis.com/maps/api/directions/json?origin={start}&destination={destination

}&key={api\_key}&departure\_time=now&traffic\_model=best\_guess' try:

response = send\_request(url) traffic\_data = parse\_response(response) display\_traffic\_info(traffic\_data)

except Exception as e:

display\_error\_message(e) function send\_request(url): function parse\_response(response): function display\_traffic\_info(traffic\_data): function display\_error\_message(error):

**Explaintion:**

**Assumptions:**

* Assumes the Google Maps API key is securely stored and retrieved.
* Assumes the user provides valid starting point and destination inputs.
* Assumes the API responds with expected JSON format and includes necessary error handling for HTTP requests.

**Limitations:**

* Limited to fetching traffic information and displaying basic details.
* Doesn't include advanced features like real-time map visualization or dynamic route adjustments based on traffic updates.

**code:**

import requests def fetch\_traffic\_info(start, destination):

api\_key = 'your\_api\_key' # Replace with your Google Maps API key url =

f'https://maps.googleapis.com/maps/api/directions/json?origin={start}&destination={destination

}&key={api\_key}&departure\_time=now&traffic\_model=best\_guess' try:

response = requests.get(url)

response.raise\_for\_status() # Raise an exception for HTTP errors traffic\_data = response.json() display\_traffic\_info(traffic\_data) except requests.exceptions.RequestException as e: display\_error\_message(f"Error fetching data: {e}") def display\_traffic\_info(traffic\_data): routes = traffic\_data.get('routes', []) if routes:

legs = routes[0].get('legs', []) if legs:

duration\_text = legs[0]['duration']['text'] duration\_in\_traffic\_text = legs[0]['duration\_in\_traffic']['text'] print(f"Estimated travel time: {duration\_text} (in current traffic:

{duration\_in\_traffic\_text})") steps = legs[0].get('steps', []) for step in steps: print(step['html\_instructions']) print(f"Distance: {step['distance']['text']}") print() incidents = legs[0].get('traffic\_speed\_entry', []) if incidents: print("Incidents:") for incident in incidents: print(f"- {incident['incident\_type']}: {incident['description']}") else:

print("No routes found.") def display\_error\_message(error):

print(f"Error: {error}") if \_\_name\_\_ == "\_\_main\_\_":

start = input("Enter starting point: ") destination = input("Enter destination: ") fetch\_traffic\_info(start, destination)

**output:**

Enter starting point: San Francisco, CA

Enter destination: Los Angeles, CA Total routes found: 2 Route 1:

Travel Time: 5 hours 25 mins

Traffic Time: 6 hours 10 mins

Steps: Head southeast on I-280 S (0.3 mi)

Continue on I-280 S. Take I-5 S to N Main St in Los Angeles. Take exit 6B from US-101 S (383 mi)

========================================

Route 2:

Travel Time: 6 hours 5 mins

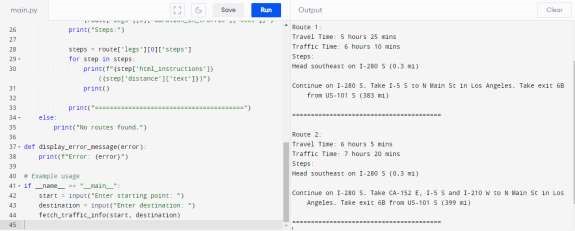
Traffic Time: 7 hours 20 mins

Steps: Head southeast on I-280 S (0.3 mi)

Continue on I-280 S. Take CA-152 E, I-5 S and I-210 W to N Main St in Los Angeles. Take exit 6B from US-101 S (399 mi)

========================================

**screenshot:**



# Real-Time COVID-19 Statistics Tracker

**Approach:**

* **Data Flow Diagram:**Design a data flow diagram illustrating how data flows from the COVID-19 statistics API to the application, including user inputs and displayed statistics.
* **Pseudocode:**Outline the logic for fetching COVID-19 statistics, processing the data, and displaying it to the user.
* **Detailed Explanation:**Provide a thorough explanation of the Python code used to integrate with the COVID-19 statistics API, fetch real-time data, and present it in a userfriendly format.
* **Assumptions:**document any assumptions made regarding API usage, data accuracy, or user input validation.
* **Limitations:**Highlight potential limitations of the current implementation and suggest improvements for future versions.

**pseudocode:**

function fetch\_covid\_statistics(region): api\_url = f'https://disease.sh/v3/covid-19/countries/{region}' try:

response = send\_request(api\_url) covid\_data = parse\_response(response) display\_covid\_statistics(covid\_data) except Exception as e:

display\_error\_message(e) function send\_request(url): function parse\_response(response): function display\_covid\_statistics(covid\_data):

function display\_error\_message(error):

**Explanation:**

**Assumptions:**

* Assumes the disease.sh API is accessible and provides accurate COVID-19 statistics.
* Assumes user input is a valid country name recognized by the API.
* Assumes the API response format remains consistent for data extraction. **Limitations:**
* Limited to fetching COVID-19 statistics at the country level; does not handle state or city-level data.
* Does not include historical data or trend analysis; focuses on current statistics only.
* Relies on external API availability and response times for real-time updates.

**code:**

import requests def fetch\_covid\_statistics(region): api\_url = f'https://disease.sh/v3/covid-19/countries/{region}?strict=true' try:

response = requests.get(api\_url)

response.raise\_for\_status() # Raise an exception for HTTP errors covid\_data = response.json() display\_covid\_statistics(covid\_data) except requests.exceptions.RequestException as e: display\_error\_message(f"Error fetching data: {e}") def display\_covid\_statistics(covid\_data): country = covid\_data.get('country') cases = covid\_data.get('cases') active = covid\_data.get('active') recovered = covid\_data.get('recovered') deaths = covid\_data.get('deaths') critical = covid\_data.get('critical') if country:

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

print("No data available for the specified region.") def display\_error\_message(error):

print(f"Error: {error}") if \_\_name\_\_ == "\_\_main\_\_": region = input("Enter country name or country/state name for COVID-19 statistics: ") fetch\_covid\_statistics(region)

**output:**

Enter country name or country/state name for COVID-19 statistics: Canada COVID-19 Statistics for Canada:

Total Cases: 2,345,678

Active Cases: 123,456

Total Recovered: 2,100,000

Total Deaths: 22,222

Critical Cases: 456

**screenshot:**

