**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.**

**Answer:**

**Model Data flow:**

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

| User |

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

|

| (1) Input location (city name or coordinates)

|

v

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

| Weather Monitoring System|

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

|

| (2) Construct API request URL with user input and API key

|

v

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

| External Weather API |

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

|

| (3) HTTP GET request to API

|

v

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

| External Weather API |

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

|

| (4) JSON response with weather data

|

v

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

| Weather Monitoring System|

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

|

| (5) Parse JSON response and extract relevant weather data

|

v

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

| User |

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

|

| (6) Display weather data to user

|

v

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

**Python Code:**

import requests

def get\_weather\_data(location):

api\_key = "b11f2fe52244a66eb93ee793f28c2d3b" # Your provided API key

base\_url = "http://api.openweathermap.org/data/2.5/weather?"

complete\_url = base\_url + "q=" + location + "&appid=" + api\_key

response = requests.get(complete\_url)

return response.json()

def display\_weather\_data(weather\_data):

if weather\_data['cod'] != '404':

main = weather\_data['main']

wind = weather\_data['wind']

weather\_description = weather\_data['weather'][0]['description']

print(f"Temperature: {main['temp']}K")

print(f"Humidity: {main['humidity']}%")

print(f"Weather Description: {weather\_description}")

print(f"Wind Speed: {wind['speed']} m/s")

else:

print("City Not Found")

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

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

weather\_data = get\_weather\_data(location)

display\_weather\_data(weather\_data)

**Pseudocode:**

**1.**Initialize the application

* Import necessary libraries
* Set up the API key and base URL for the weather API

**2.**Get user input

* Prompt the user to input a location (city name or coordinates)

**3.**Fetch weather data

* Build the request URL using the user input and API key
* Make an HTTP GET request to the weather API
* Parse the JSON response to extract relevant weather data

**4.**Display weather information

* Format and display the current weather information: temperature, weather conditions, humidity, and wind speed

**5.**Error handling

* Handle any errors that may occur during the API request or data parsing

**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.**

**Answer:**

**Model Data flow:**

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

| Sales Data |

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

|

| (1) Sales Transactions

v

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

| Inventory Management System |

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

|

| (2) Update Stock Levels

|

v

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

| Database |

| - Products |

| - Warehouses |

| - Current Stock Levels |

| - Historical Sales Data |

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

|

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

| |

v v

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

| Inventory Tracking Module | | Inventory Optimization |

+-----------------------------+ | Module |

| +---------------------------+

| | |

| | (4) Calculate Reorder Points|

v | and Quantities |

+-----------------------------+ | |

| User Interface (UI) | v |

| - Input Product IDs/Names | <------- +---------------------------+

| - View Current Stock Levels | | Reporting Module |

| - View Reorder Recommendations | (3) Reorder | - Generate Inventory |

| - View Historical Data | Alerts | Turnover Reports |

| - View Reports | | - Generate Stockout |

+-----------------------------+ | Occurrences Reports |

| - Generate Overstock Cost |

| Reports |

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

**Python Code:**

class Product:

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

self.product\_id = product\_id

self.name = name

self.category = category

self.current\_stock = current\_stock

self.reorder\_point = reorder\_point

self.reorder\_quantity = reorder\_quantity

self.lead\_time = lead\_time

self.historical\_sales = []

def add\_sales\_data(self, sales):

self.historical\_sales.append(sales)

self.current\_stock -= sales # Update current stock after sale

class Warehouse:

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

self.warehouse\_id = warehouse\_id

self.location = location

self.products = {}

def add\_product(self, product):

self.products[product.product\_id] = product

def track\_inventory(self):

print("\n--- Inventory Tracking ---")

for product in self.products.values():

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

print(f"Reorder Alert for: {product.name}")

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

print(f"Recommended Order Quantity: {product.reorder\_quantity}")

else:

print(f"{product.name} is sufficiently stocked.")

def generate\_report(self):

print("\n--- Inventory Report ---")

for product in self.products.values():

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

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

print(f"Turnover Rate: {self.calculate\_turnover\_rate(product)}")

def calculate\_turnover\_rate(self, product):

total\_sales = sum(product.historical\_sales)

average\_stock = (product.current\_stock + product.current\_stock) / 2

turnover\_rate = total\_sales / average\_stock if average\_stock > 0 else 0

return turnover\_rate

def calculate\_eoq(annual\_demand, ordering\_cost, holding\_cost):

if holding\_cost > 0:

return (2 \* annual\_demand \* ordering\_cost / holding\_cost) \*\* 0.5

return 0

def main():

warehouse = Warehouse(1, "Main Warehouse")

# Example products

product1 = Product(101, "Laptop", "Electronics", 15, 5, 20, 2)

product1.add\_sales\_data(3)

product1.add\_sales\_data(4)

product2 = Product(102, "Smartphone", "Electronics", 30, 10, 15, 3)

product2.add\_sales\_data(5)

product2.add\_sales\_data(6)

warehouse.add\_product(product1)

warehouse.add\_product(product2)

while True:

print("\nOptions:")

print("1. Track Inventory")

print("2. Generate Report")

print("3. Add Sales Data")

print("4. Calculate EOQ")

print("5. Exit")

choice = input("Select an option: ")

if choice == '1':

warehouse.track\_inventory()

elif choice == '2':

warehouse.generate\_report()

elif choice == '3':

product\_id = int(input("Enter Product ID: "))

sales = int(input("Enter sales data: "))

if product\_id in warehouse.products:

warehouse.products[product\_id].add\_sales\_data(sales)

print("Sales data updated.")

else:

print("Product not found.")

elif choice == '4':

product\_id = int(input("Enter Product ID: "))

if product\_id in warehouse.products:

annual\_demand = sum(warehouse.products[product\_id].historical\_sales)

ordering\_cost = 50 # Example ordering cost

holding\_cost = 2 # Example holding cost per unit

eoq = calculate\_eoq(annual\_demand, ordering\_cost, holding\_cost)

print(f"Optimal Order Quantity (EOQ) for {warehouse.products[product\_id].name}: {eoq:.2f}")

else:

print("Product not found.")

elif choice == '5':

break

else:

print("Invalid option, try again.")

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

main()

**Pseudocode:**

1. Define class Product:

* Initialize with product\_id, name, current\_stock
* Initialize empty list for historical\_sales

2. Define class Warehouse:

* + Initialize with warehouse\_id, name
  + Initialize empty dictionary for inventory

3. Define class InventorySystem:

* + Initialize with empty dictionaries for products and warehouses

4.Define method add\_product(product):

* + Add product to products dictionary

5.Define method add\_warehouse(warehouse):

* Add warehouse to warehouses dictionary

6.Define method update\_stock(product\_id, warehouse\_id, quantity):

* + Retrieve warehouse by warehouse\_id
  + Update inventory for product\_id in warehouse
  + Retrieve product by product\_id
  + Update product current\_stock

7.Define method record\_sale(product\_id, quantity):

* Retrieve product by product\_id
  + Decrease product current\_stock by quantity

**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.**

**Answer:**

**Data Flow Model:**

Start

|

V

[User Input: Starting Point, Destination]

|

V

[Send Request to Traffic API]

|

V

[Receive Traffic Data]

|

V

[Process Traffic Data]

|

V

[Display Traffic Conditions, Estimated Travel Time, Incidents]

|

V

[Optionally Display Alternative Routes]

|

V

End

**Python code:**

import requests

API\_KEY = 'YOUR\_GOOGLE\_MAPS\_API\_KEY'

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

def fetch\_traffic\_data(start, destination):

params = {

'origin': start,

'destination': destination,

'key': API\_KEY,

'departure\_time': 'now'

}

response = requests.get(BASE\_URL, params=params)

data = response.json()

if data['status'] == 'OK':

route = data['routes'][0]

legs = route['legs'][0]

traffic\_data = {

'traffic\_conditions': legs['traffic\_speed\_entry'],

'travel\_time': legs['duration\_in\_traffic']['text'],

'incidents': route['warnings']

}

return traffic\_data

else:

raise Exception('Error fetching traffic data: ' + data['status'])

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

print("Current Traffic Conditions: ")

for condition in traffic\_data['traffic\_conditions']:

print(f" - Speed: {condition['speed']} km/h")

print(f"Estimated Travel Time: {traffic\_data['travel\_time']}")

print("Incidents or Delays: ")

if traffic\_data['incidents']:

for incident in traffic\_data['incidents']:

print(f" - {incident}")

else:

print(" - No incidents or delays reported.")

def main():

start = input("Enter the starting point: ")

destination = input("Enter the destination: ")

try:

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

display\_traffic\_data(traffic\_data)

except Exception as e:

print(f"An error occurred: {e}")

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

main()

**Pseudocode:**

1. Define constants for the API key and base URL of the traffic monitoring API.

2. Create a function `fetch\_traffic\_data(start, destination)`:

* Construct the API request URL with the start and destination points.
* Send a request to the API and get the response.
* Parse the response to extract traffic data (conditions, travel time, incidents).
* Return the extracted traffic data.

3. Create a function `display\_traffic\_data(traffic\_data)`:

* Print current traffic conditions.
* Print estimated travel time.
* Print any incidents or delays.
* Suggest alternative routes if traffic is heavy.

4. Create a function `main()`:

* Prompt the user for a starting point and destination.
* b. Call `fetch\_traffic\_data(start, destination)` to get real-time traffic data.
* c. Call `display\_traffic\_data(traffic\_data)` to display the information.

5. Execute the `main()` function.

**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.**

**Answer :**

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

| User |

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

|

| (1) Input region (country, state, or city)

v

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

| COVID-19 Statistics Tracker|

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

|

| (2) Construct API request URL with user input

|

v

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

| External COVID-19 API |

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

|

| (3) HTTP GET request to API

|

v

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

| External COVID-19 API |

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

|

| (4) JSON response with COVID-19 statistics

|

v

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

| COVID-19 Statistics Tracker|

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

|

| (5) Parse JSON response and extract relevant data

|

v

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

| User |

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

|

| (6) Display COVID-19 statistics to user

|

v

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

**Python Code:**

import requests

def fetch\_covid\_data(region):

API\_URL = f"https://disease.sh/v3/covid-19/countries/{region}"

response = make\_api\_call(API\_URL)

if response.status\_code == 200:

return response.json()

else:

return f"Error fetching data: {response.status\_code}"

def make\_api\_call(url):

headers = {"Accept": "application/json"}

return requests.get(url, headers=headers)

def display\_statistics(data):

print(f"COVID-19 Statistics for {data['country']}:")

print(f"Total Cases: {data['cases']}")

print(f"Total Recoveries: {data['recovered']}")

print(f"Total Deaths: {data['deaths']}")

def main():

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

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

if isinstance(covid\_data, dict):

display\_statistics(covid\_data)

else:

print(covid\_data)

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

main()

**Pseudocode:**

**1.** Define class CovidStatsTracker:

* + Initialize with api\_key and base\_url
  + Define method get\_covid\_stats(region):

**2.**Construct request URL using base\_url, region, and API key

* + Send HTTP GET request to the API
  + If response is successful:
  + Parse JSON response
  + Extract current cases, recoveries, and deaths
  + Return extracted data

- Else:

* Return None

**3.**Define method display\_stats(data):

- If data is not None:

* Print current cases, recoveries, and deaths

- Else:

* + Print error message