**Assignment: Python Programming for GUI Development**

Name: K.Prathiksha

Register Number: 192321173

Department: IT

Date of Submission: 29.8.2024

**Problem 1:** **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.

# Solution:

# Real-Time Weather Monitoring System

# 1.Data Flow Diagram

pasted-image.tiff

# 2. Implementation

|  |
| --- |
| inventory = {}  def add\_item(name, reorder\_level):  inventory[name] = {'stock': 0, 'reorder\_level': reorder\_level}  def update\_inventory(name, quantity, transaction\_type):  if transaction\_type == "purchase":  inventory[name]['stock'] += quantity  elif transaction\_type == "sale":  inventory[name]['stock'] -= quantity  def check\_and\_reorder():  for name, details in inventory.items():  if details['stock'] < details['reorder\_level']:  print(f"Reorder needed for {name}: Current stock = {details['stock']}")  # Example usage  add\_item("Item A", 17)  update\_inventory("Item A", 20, "purchase")  update\_inventory("Item A", 5, "sale")  add\_item("Item B",20)  update\_inventory("Item B", 20, "purchase")  update\_inventory("Item B", 7, "sale")  check\_and\_reorder() |

# 3.Display the Current weather information

Number of items A:17

Inventory purchase: 20

Sale:5

Number of items B: 20

Inventory purchase: 20

Sale: 7

# 4.User Inputpasted-image.tiff

**5.Documentation**

**Algorithms**

Inventory management systems use reorder optimization algorithms to ensure that stock levels are maintained efficiently, preventing both stockouts and overstock situations.They help balance trade-offs between holding costs, ordering costs, stockouts, and overstocks. They can also consider other factors like lead times, service levels, discounts, and promotions.

Data Structure Initialization

Adding an Item to Inventory

Updating the Inventory

Checking and Reordering

Displaying Inventory Information

Displaying the inventory information for all items.

**3. Assumptions Made in Inventory Management System Optimization**

Inventory management systems often operate under several assumptions to streamline processes. These assumptions include:

**a. Constant Lead Times:**

* **Assumption:** The time between placing an order and receiving the goods remains constant for each supplier.
* **Implications:** In practice, lead times may vary due to unforeseen issues (e.g., supplier delays, shipping problems). To mitigate risks, systems incorporate safety stock, but this assumption simplifies calculations.

**b. Constant Demand Rate:**

* **Assumption:** The system often assumes a constant demand rate for each product, based on historical averages.
* **Implications:** Variability in demand can challenge this assumption, but it helps simplify reorder point and EOQ calculations. Systems account for fluctuations through demand forecasting models.

**c. Infinite Supply Availability:**

* **Assumption:** Suppliers can always fulfill orders regardless of size or timing.
* **Implications:** This is not always true in practice. In case of shortages, systems may need to prioritize orders for essential or high-priority items.

**d. No Interdependence Between Items:**

* **Assumption:** Inventory items are managed independently.
* **Implications:** In reality, there may be dependencies, such as bundled products or components that are used together in manufacturing. Systems need to account for these links through advanced algorithms.

**e. Static Costs:**

* **Assumption:** Ordering costs, holding costs, and shortage costs remain constant over time.
* **Implications:** Costs may fluctuate due to market conditions, inflation, or changes in supplier agreements. Systems can adapt by recalculating EOQ and reorder points regularly.

.

**Conclusion**

Reorder optimization algorithms, influenced by historical data, aim to ensure efficient inventory management by balancing costs, demand, and supply constraints. Assumptions such as constant lead times and demand rates simplify the system but may need periodic adjustment to account for real-world variability. These algorithms are key to maintaining optimal stock levels and avoiding costly stockouts or excess inventory.