1. Problem Statement: Employee Attendance Management System

Context:

You are working for a mid-sized company that has recently grown and now has multiple departments and a significant number of employees. The company needs a system to manage and track employee attendance. The HR department has been doing this manually using spreadsheets, but as the company grows, this method has become inefficient and prone to errors.

Objective:

Develop a Python-based Employee Attendance Management System that reads, updates, and manages employee attendance records using file handling.

Requirements:

1. **Data Storage:**
    - Employee data, including their ID, name, department, and daily attendance records, should be stored in text files. Each employee should have a unique file named using their employee ID (e.g., `emp123.txt`).
    - The file should store daily attendance logs in the format: `Date, Check-in Time, Check-out Time, Hours Worked`.
2. **Features:**
    - **Check-in/Check-out:**
        - Allow employees to check in and check out. The system should log the current date, time of check-in, and check-out, and calculate the hours worked for the day.
    - **Add New Employee:**
        - Provide a function to add a new employee to the system. This should create a new file for the employee.
    - **Update Employee Information:**
        - Allow updates to employee information, such as name or department.
    - **Generate Attendance Report:**
        - Generate a monthly attendance report for each employee that summarizes the total hours worked each day and overall for the month. This report should be saved in a separate file (e.g., `emp123_report.txt`).
3. **Constraints:**
    - The system should handle file errors gracefully, such as trying to access a file that does not exist.
    - Ensure data integrity by preventing duplicate entries for the same day.
    - Implement basic input validation to avoid invalid data entries (e.g., incorrect date format, check-out before check-in).

Example Workflow:

1. An employee checks in at 9:00 AM. The system logs the check-in time.
2. At 5:00 PM, the employee checks out. The system logs the check-out time and calculates the hours worked (e.g., 8 hours).
3. At the end of the month, HR generates a report showing the employee's daily attendance and total hours worked.

Deliverables:

- Python script(s) implementing the system.
- Text files representing employee records and attendance logs.
- A report for at least one employee showing a month's worth of attendance data.

---

2. Problem Statement: Restaurant Order and Inventory Management System

Context:

You are working with a small restaurant that has been managing its orders and inventory manually, using paper logs and spreadsheets. As the restaurant gains popularity, managing these tasks manually has become time-consuming and error-prone. The restaurant needs a system that can handle orders, update inventory levels, and generate reports on stock usage and sales.

Objective:

Develop a Python-based Restaurant Order and Inventory Management System that uses file handling to manage order records, inventory levels, and sales reports.

Requirements:

1. **Data Storage:**
    - **Menu Items:** Store the restaurant's menu in a text file (`menu.txt`). Each line should contain details about a menu item, including `Item ID`, `Name`, `Price`, and `Stock Quantity`.
    - **Orders:** Each order should be recorded in a separate text file (`order_<order_id>.txt`), containing the details of the items ordered, their quantities, and the total price.
    - **Inventory:** Maintain an inventory file (`inventory.txt`) to track the quantity of ingredients available. Each line should contain `Ingredient ID`, `Name`, `Current Stock`, and `Minimum Required Stock`.
2. **Features:**
    - **Place Order:**
        - Allow the restaurant staff to place an order by selecting menu items and quantities. The system should check if there is sufficient stock before confirming the order.
        - Deduct the used ingredients from the inventory file based on the order.

        - Save the order details in a separate file, including the date, time, and total price.
    - **Add/Update Menu Items:**
        - Provide functionality to add new menu items or update existing ones, including price and stock quantity.
    - **Inventory Management:**
        - Automatically update the inventory when an order is placed.
        - Provide a feature to manually update inventory levels (e.g., after receiving a delivery).
        - Generate a warning or report if any ingredient falls below the minimum required stock.
    - **Generate Sales Report:**
        - Generate a daily or weekly sales report summarizing the total revenue, the number of each menu item sold, and the current inventory status. The report should be saved as a separate file (`sales_report_<date>.txt`).

3. **Constraints:**

    - The system should handle file errors gracefully, such as trying to access a file that does not exist or writing to a file with insufficient permissions.
    - Prevent orders from being placed if there isn't enough stock for any of the requested items, and notify the staff which items are unavailable.
    - Ensure that the menu file and inventory file are always synchronized when menu items or stock levels are updated.

Example Workflow:

1. A customer places an order for 2 burgers and 1 soda. The system checks if the restaurant has enough stock of the necessary ingredients.
2. The order is confirmed, and the ingredients used (e.g., buns, patties, soda cans) are deducted from the inventory.
3. The system saves the order details in a new file and updates the sales report for the day.
4. The inventory file is updated. If the stock of buns falls below the minimum required level, a warning is generated.

Deliverables:

- Python script(s) implementing the system.
- Text files representing the menu, inventory, orders, and sales reports.
- At least one example order and sales report file.

---

3. Problem Statement: Real Estate Property Management and Analytics System

Context:

You are working with a real estate company that manages a large portfolio of properties including residential, commercial, and rental units. The company currently tracks property information, tenant details, rent

payments, and maintenance requests manually using spreadsheets, which has become inefficient and prone to errors as the business scales.

Objective:

Develop a Python-based Real Estate Property Management and Analytics System that uses CSV files to manage property details, tenant information, rent payments, and maintenance logs. The system should also generate insightful analytics reports to help the company make informed decisions.

Requirements:

1. **Data Storage:**
    - **Properties:** Store property details in a CSV file (`properties.csv`). Each row should include `Property ID`, `Address`, `Type` (residential/commercial), `Size (sq ft)`, `Rental Price`, `Status` (available/occupied), and `Owner Name`.
    - **Tenants:** Maintain a CSV file for tenants (`tenants.csv`) with details such as `Tenant ID`, `Name`, `Property ID`, `Lease Start Date`, `Lease End Date`, and `Monthly Rent`.
    - **Rent Payments:** Track rent payments in a CSV file (`rent_payments.csv`) with columns for `Payment ID`, `Tenant ID`, `Property ID`, `Payment Date`, `Amount Paid`, and `Payment Status` (on-time/late).
    - **Maintenance Requests:** Keep maintenance requests in a CSV file (`maintenance.csv`) including `Request ID`, `Property ID`, `Issue Description`, `Request Date`, `Status` (pending/in-progress/completed), and `Cost`.
2. **Features:**
    - **Add/Update Properties:**
        - Add new properties or update existing property details, including changing the status when a property is rented out or becomes available.
    - **Manage Tenants:**
        - Add new tenants and update their lease details. Link tenants to properties and automatically update the property's status when a new tenant is added.
    - **Process Rent Payments:**
        - Record rent payments, update payment status (on-time or late based on the payment date), and generate payment receipts.
        - Alert the management team if a tenant misses a payment or if the payment is late.
    - **Manage Maintenance Requests:**
        - Log new maintenance requests, update their status as work progresses, and record the costs associated with each request.
        - Generate a maintenance cost report for each property and a summary report of all maintenance requests in a given period.
    - **Analytics and Reports:**
        - **Occupancy Rate Report:** Generate a report showing the current occupancy rate for all properties.
        - **Revenue Report:** Create monthly or yearly revenue reports based on rent payments received, highlighting total revenue, outstanding payments, and late payments.

        - **Property Performance Analysis:** Analyze properties to identify those with the highest and lowest occupancy rates, highest maintenance costs, and best revenue generation.

3. **Constraints:**

    - Ensure data integrity and prevent duplicates by validating data entries (e.g., no two properties should have the same Property ID).
    - Implement error handling for file operations, such as checking if the CSV files exist and managing read/write permissions.
    - Handle edge cases, such as lease end dates that have passed, missing payment records, or maintenance requests that are left unresolved for too long.
    - Allow the system to be extended easily with new features, such as tracking property upgrades or handling tenant move-out processes.

Example Workflow:

1. A new property is added to the system with all its details. When a tenant signs a lease for the property, the tenant's details are linked, and the property status is updated to "occupied."
2. Each month, tenants make rent payments. The system records these payments, checks if they are on-time or late, and updates the rent payment status.
3. A maintenance issue is reported for a property. The request is logged, and its status is updated as work progresses. Once completed, the cost is recorded, and the system updates the property's maintenance cost report.
4. At the end of each month, the system generates an occupancy rate report and a revenue report, helping the company assess property performance and financial health.

Deliverables:

- Python script(s) implementing the system with functionalities as described.
- CSV files representing properties, tenants, rent payments, and maintenance logs.
- Example reports including occupancy rates, revenue summaries, and property performance analytics.
