# Problem 1

### E-Commerce Order Processing System - Detailed Problem Statement

Developing a basic E-Commerce Order Processing System as part of its digital transformation initiatives. The objective of this application is to manage customer order details using core Python data structures and modular programming concepts.

### Problem Statement

As part of an e-commerce company's digital transformation initiative, you are required to develop an Order Processing System. The application should store multiple customer order records, use Python data structures to manage the data, and allow filtering of orders based on product type.

### Apply user defined functions

| Task No | Task Name | Activity Description | Expected Result |
|---|---|---|---|
| Task 1 | Store Order Records | Capture and store multiple orders | Order list created |
| Task 2 | Maintain Product Lookup | Store product details using dictionary | Product info available |
| Task 3 | Filter Orders by Product | Display orders for a given product | Filtered order list |
| Task 4 | Function call | Modularize logic using functions | Structured program |

# Task 1: Store Order Records

Capture details of multiple e-commerce orders.

**Steps to Do**

1. Create a module - `orders.order_records`
2. Creat `store_order_records` as a function
3. Accept Customer Name
4. Accept Product Type (Electronics, Clothing, Books)
5. Accept Order Amount
6. Store each order record as a tuple
7. Store all order records in a list

**Expected Result** Multiple order records are stored successfully.

# Task 2: Maintain Product Lookup

Use a dictionary to store and retrieve product information.

**Steps to Do**

1. Create a module `products.product_lookup`
2. Create function `maintain_product_lookup`
3. Create a dictionary for product details
4. Store product descriptions for Electronics, Clothing, Books
5. Use dictionary lookup to fetch product information

```
product_lookup = {
    "Electronics": "Electronics – Gadgets & Devices",
    "Clothing": "Clothing – Apparel & Fashion",
    "Books": "Books – Literature & Education"
}
```

**Expected Result** Correct product details are retrieved using dictionary lookup.

# Task 3: Filter Orders by Product

Display orders belonging to a specific product type.

**Steps to Do**

1. Create a module `display.orders`
2. Create function - `display_orders function`
3. Create a module - `utils.filters`
4. Create function - `filter_orders_by_product`
5. Accept a product type from the user
6. Search order records for matching product
7. Display matching orders along with product details
8. Display a message if no orders are found

**Expected Result** Filtered order details are displayed correctly.

# Task 4: Function call

Improve code structure using functions.

**Steps to Do**

1. Call a function to display order details
2. Call a function for Product lookup
3. Call a function to display filtered orders by product

**Expected Result** Program is modular and easy to maintain.

**Sample Input & Output**

```
Enter number of orders: 3
Order 1
Customer: Hiroshi
Product: Electronics
Amount: 25000

Order 2
Customer: Kenji
Product: Clothing
Amount: 4500

Order 3
Customer: Takeshi
Product: Electronics
Amount: 23000
```

```
All Orders:
----------------------------------
Customer: ('Hiroshi', 'Electronics', 25000.0) Product: ('Hiroshi',
'Electronics', 25000.0) Amount: ('Hiroshi', 'Electronics',
25000.0)
Customer: ('Kenji', 'Clothing', 4500.0) Product: ('Kenji',
'Clothing', 4500.0) Amount: ('Kenji', 'Clothing', 4500.0)
Customer: ('Takeshi', 'Electronics', 23000.0) Product: ('Takeshi',
'Electronics', 23000.0) Amount: ('Takeshi', 'Electronics',
23000.0)
----------------------------------
Enter product type to filter: Electronics

Orders in Electronics – Gadgets & Devices
----------------------------------
Customer: Hiroshi Amount: 25000.0
Customer: Takeshi Amount: 23000.0
----------------------------------
```

# Problem 2

### Healthcare Patient Appointment Management System - Detailed Problem Statement (Advanced)

Developing an advanced Healthcare Patient Appointment Management System using Python data structures and modular programming with additional validation and summary features.

### Problem Statement

As part of a hospital's digital transformation initiative, develop a Patient Appointment Management System. The system stores multiple patient appointment records, maintains department lookups, filters by department, validates appointment times, and provides summary statistics. Use modular functions with error handling.

### Apply user defined functions

| Task No | Task Name | Activity Description | Expected Result |
|---|---|---|---|
| Task 1 | Store Patient Records | Capture/validate multiple appointments | Validated appointment list |
| Task 2 | Maintain Department Lookup | Store department details in dictionary | Department info available |
| Task 3 | Filter Appointments by Dept | Display appointments for given department | Filtered appointment list |
| Task 4 | Generate Summary Statistics | Count appointments by department | Summary report generated |
| Task 5 | Modular Function Calls | Orchestrate all functions | Structured, robust program |

# Task 1: Store Patient Records (with Validation)

Capture validated patient appointment details.

**Steps to Do**

1. Create modules `patients.patient_records` & `utils.time_utils`
2. Create Function inside `utils.time_utils` to Validate HH:MM format and business hours
3. Use `utils.time_utils` function in our `store_patient_records` function
4. Create function `store_patient_records`
5. Accept Patient Name (non-empty)
6. Accept Department (Cardiology, Orthopedics, Neurology)
7. Accept Appointment Time (HH:MM format, 09:00-18:00)
8. Validate time format and range
9. Store as tuple: (name, dept, time)
10. Handle invalid inputs with retry

**Expected Result** Only valid appointment records stored.[^1]

# Task 2: Maintain Department Lookup

Dictionary for department details.

**Steps to Do**

1. Create module `department.department_lookup`
2. Create function returning dictionary
3. Store: Cardiology/Cardiology-Heart Care, Orthopedics/Bone & Joint Care, Neurology/Brain & Nerve Care
4. Include doctor count per department

**Expected Result** Department details accessible via lookup.

# Task 3: Filter Appointments by Department

Display department-specific appointments.

**Steps to Do**

1. Create module `utils.filters`
2. Create filtering function `filter_appointments_by_dept`
3. Accept department input
4. Case-insensitive matching
5. Sort by appointment time
6. Show “No appointments” if empty

**Expected Result** Sorted, filtered appointments displayed.

# Task 4: Generate Summary Statistics

Department-wise appointment counts.

**Steps to Do**

1. Create summary function
2. Count appointments per department
3. Display busiest department

4. Total appointments count

**Expected Result** Statistical summary generated.

# Task 5: Modular Execution

Orchestrate complete workflow.

### Steps to Do

1. Sequential function calls
2. Error handling wrapper
3. Clean menu-driven interface

**Expected Result** Production-ready modular application.

### Sample Input/Output

```
Enter number of appointments: 2

Patient 1:
Name: Chiyoko
Department (Cardiology/Orthopedics/Neurology): abc
Invalid department!
Department (Cardiology/Orthopedics/Neurology): Cardiology
Time (HH:MM): aa
Invalid time! Use HH:MM (09:00–18:00)
Time (HH:MM): 08:00
Invalid time! Use HH:MM (09:00–18:00)
Time (HH:MM): 09:15

Patient 2:
Name: Kiyoko
Department (Cardiology/Orthopedics/Neurology): Orthopedics
Time (HH:MM): 12:00

All Appointments:
----------------------------------
09:15 | Cardiology–Heart Care (Dr. Patel) | Chiyoko
12:00 | Orthopedics–Bone & Joint Care (Dr. Singh) | Kiyoko
----------------------------------
```

```
Enter department to filter: Orthopedics

Orthopedics-Bone & Joint Care (Dr. Singh) Appointments (sorted):
----------------------------------
12:00 - Kiyoko
----------------------------------

Summary Statistics:
Cardiology: 1
Orthopedics: 1
Cardiology: 1 (Busiest)
Total: 2
```