Python Data Engineering Coding Exercises

These exercises are designed to help developers practice essential Python concepts commonly used in data engineering. Each exercise presents a real-world scenario and includes multiple tasks that build upon core Python concepts including collections, lambda functions, data filtering, and JSON processing.

Exercise 1: Log File Analysis System

Scenario

You're working at a cloud services company that needs to analyze system logs from multiple servers. The logs are in JSON format and contain information about server events, errors, and performance metrics.

Task

Create a system that can process and analyze these log entries to help identify potential issues and generate reports.

Sample Input Data

```
[
    {
        "timestamp": "2024-01-15T10:30:00",
        "server_id": "srv-001",
        "event_type": "performance",
        "metrics": {
            "cpu_usage": 85.5,
            "memory_usage": 90.2,
            "disk_usage": 72.8
        },
        "status": "warning",
        "priority": "high"
    }
]
```

Requirements

1. Create a function that filters log entries to find:
    - All high-priority warnings using filter() and lambda
    - Servers with CPU usage above 80% using list comprehension
2. Create a function that extracts unique server IDs using map() and set()
3. Create a function that sorts the log entries by:
    - Timestamp (primary key)
    - Priority (secondary key) Using the sorted() function with a lambda key
4. Bonus: Create a function that generates a summary report showing:
    - Count of events by priority
    - List of unique event types

- Average CPU usage across all servers

Exercise 2: Data Schema Validator

Scenario

You're building a data validation system for a data warehouse. The system needs to validate incoming data against predefined schemas stored in JSON format.

Task

Create a schema validation system that can process incoming data and verify it matches the expected structure and data types.

Sample Schema JSON

```
{
    "tables": [
        {
            "name": "customers",
            "columns": [
                {
                    "name": "customer_id",
                    "type": "string",
                    "required": true,
                    "validation": {
                        "pattern": "^CUS[0-9]{6}$"
                    }
                },
                {
                    "name": "purchase_amount",
                    "type": "decimal",
                    "required": true,
                    "validation": {
                        "min": 0,
                        "max": 1000000
                    }
                }
            ]
        }
    ]
}
```

Requirements

1. Create a function that extracts column details from the schema file:
    - Use lambda functions to transform complex column definitions
    - Create a mapping of table names to their column specifications
2. Create a validation function that:
    - Filters required columns using filter() and lambda
    - Validates data types and constraints

- Returns validation errors in a structured format

3. Create a function that sorts validation errors by:
    - Error severity
    - Table name
    - Column name

Exercise 3: ETL Pipeline for E-commerce Analytics

Scenario

You're building an ETL pipeline for an e-commerce platform that needs to process daily transaction data and generate analytics reports.

Task

Create a system that can process raw transaction data, transform it into analytical formats, and generate summary reports.

Sample Transaction Data

```
[
    {
        "transaction_id": "T123456",
        "timestamp": "2024-01-15T14:30:00",
        "customer": {
            "id": "CUS123",
            "region": "North"
        },
        "items": [
            {
                "product_id": "P789",
                "category": "Electronics",
                "price": 499.99,
                "quantity": 1
            },
            {
                "product_id": "P456",
                "category": "Accessories",
                "price": 29.99,
                "quantity": 2
            }
        ],
        "payment_method": "credit_card",
        "status": "completed"
    }
]
```

Requirements

1. Create a function that transforms raw transaction data:
    - Flatten nested JSON structures using lambda functions

    - Calculate total transaction values
    - Extract unique product categories using map() and set()
2. Create analysis functions that:
    - Group transactions by region and calculate regional sales
    - Find top-selling products using sorted() with custom key
    - Calculate average transaction value by payment method
3. Create a report generation function that:
    - Filters completed transactions using filter()
    - Sorts data by multiple criteria using lambda
    - Generates summary statistics for different time periods

Each exercise builds upon the previous ones, gradually introducing more complex scenarios while reinforcing core concepts. The examples use realistic data structures and common business requirements that developers are likely to encounter in their data engineering careers.