Procedural Approach:

Code Structure:

Written in C, and maintains the procedural paradigm.

Uses structs (Product, ProductStock, Shop, Customer) to organize data.

Data Handling:

Uses arrays and structures for data management.

File handling is done using standard C file I/O functions.

Memory Management:

Memory is managed explicitly with dynamic allocation (malloc and free).

Structs and arrays are used for storing data.

Error Handling:

Error handling is done using standard C error reporting (fprintf(stderr, ...)).

Memory allocation errors are checked.

Data Class Approach (with Procedural Touch):

Code Structure:

Written in Python and utilizes data classes.

Adds a procedural touch for data manipulation within functions.

Data Handling:

Utilizes data classes (Product, ProductStock, Shop, Customer) for organizing data.

CSV file reading is done using the csv module in a procedural way.

Memory Management:

Memory management is abstracted by Python and is not explicitly handled.

The decimal module is used for precise handling of decimal numbers.

Error Handling:

Uses Python's exception handling for error scenarios.

Reports errors with descriptive messages.

Common Points:

Both versions support reading data from CSV files.

Both versions have a mechanism to process orders, calculate costs, and handle customer budgets.

Live mode is supported for interactive input.

Differences:

Language:

Procedural: C

Data Class (with Procedural Touch): Python

Memory Management:

Procedural: Explicit memory allocation and deallocation in C.

Data Class (with Procedural Touch): Python's garbage collector handles memory.

Error Handling:

Procedural: Relies on standard C error handling with fprintf(stderr, ...).

Data Class (with Procedural Touch): Uses Python's exception handling, providing more descriptive error messages.

File Handling:

Procedural: Uses standard C file I/O functions.

Data Class (with Procedural Touch): Utilizes Python's csv module.

Overall:

The data class approach in Python provides a more modern and concise way of organizing data.

The procedural touch is added to the data class version for better readability and maintenance.

Python's built-in features, like data classes, make the code more expressive and less error-prone

Shop Management System: Procedural vs. Object-Oriented Approach in Python and C

1. Introduction

This report presents a comparison between the procedural and object-oriented implementations of a Shop Management System in Python and C. The procedural style is demonstrated through the use of dictionaries in Python and structs in C, while the object-oriented approach is exclusively implemented in Python. The evaluation focuses on code structure, data handling, memory management, error handling, and overall design principles.

2. Procedural Approach

2.1 Python with Dictionaries

2.1.1 Code Structure

In the procedural Python implementation, dictionaries are utilized to store product information and manage the shop's stock. Functions encapsulate specific tasks, such as processing orders and calculating costs.

2.1.2 Data Handling

Data is organized using dictionaries, providing a straightforward structure to manage product information and quantities. Functions manipulate these dictionaries to simulate shop transactions.

2.1.3 Memory Management

Python's memory management system handles memory implicitly. Dictionaries are created and modified without explicit allocation or deallocation, simplifying memory management.

2.1.4 Error Handling

Error handling relies on standard Python exception mechanisms. Functions raise exceptions on encountering errors, providing basic error reporting.

2.2 C with Structs

2.2.1 Code Structure

The procedural C implementation uses structs to define the data structure. Functions are employed to process orders, calculate costs, and manage the shop's stock.

2.2.2 Data Handling

Data is organized using structs, encapsulating product information and quantities. Functions manipulate these structs to simulate shop transactions.

2.2.3 Memory Management

Memory management is explicit in C. Memory must be manually allocated and deallocated for structs, introducing complexities compared to Python's implicit memory management.

2.2.4 Error Handling

Error handling in C relies on manual checks and return values. Functions return error codes or flags to indicate issues, requiring careful programming to handle errors effectively.

3. Object-Oriented Approach (Python)

3.1 Code Structure

In the object-oriented Python implementation, classes (Product, ProductStock, Shop, Customer) are used to encapsulate data and behavior. Objects are instantiated from these classes, promoting a modular and organized structure.

3.2 Data Handling

Data is encapsulated within class instances, enhancing code organization and separation of concerns. Methods within classes handle operations related to the specific class, promoting code encapsulation.

3.3 Memory Management

Memory management is handled implicitly by Python's garbage collector. Instances of classes are created and destroyed as needed, reducing the risk of memory leaks.

3.4 Error Handling

Error handling is integrated into class methods, providing a more localized and context-aware approach. Exceptions are raised and caught within methods, improving code readability.

4. Comparison

4.1 Common Points

Both procedural and object-oriented approaches achieve the required functionality, including reading data from CSV files, processing orders, calculating costs, and handling customer budgets.

Live mode, allowing interactive input, is available in all implementations.

4.2 Differences

Code Structure:

Procedural (Python): Dictates the use of dictionaries and functions.

Procedural (C): Relies on structs and functions.

Object-Oriented (Python): Utilizes classes and methods.

Data Handling:

Procedural (Python): Uses dictionaries for data storage.

Procedural (C): Employs structs for data organization.

Object-Oriented (Python): Encapsulates data within class instances.

Memory Management:

Procedural (Python): Implicit memory management.

Procedural (C): Explicit memory management.

Object-Oriented (Python): Implicit memory management with garbage collection.

Error Handling:

Procedural (Python): Leverages standard Python exceptions.

Procedural (C): Requires manual error checks and return values.

Object-Oriented (Python): Integrates error handling within class methods.

5. Overall Evaluation

The procedural implementations in both Python and C demonstrate functional shop management systems. However, the object-oriented Python implementation offers a more modular, organized, and maintainable code structure. The use of classes and encapsulation aligns with object-oriented design principles, making the code extensible and easier to comprehend.

For smaller systems or projects with less complexity, the procedural approach in either Python or C may suffice. However, for larger systems or those with potential future expansions, the object-oriented approach in Python is recommended due to its superior code organization and design principles.

6. Conclusion

This report has compared the procedural and object-oriented implementations of a Shop Management System in Python and C. The analysis considered various aspects, highlighting the strengths and weaknesses of each approach. The choice between procedural and object-oriented styles should be made based on the specific requirements and future scalability of the system.