SHOPPING CART SIMULATOR

Course Project Submission for CSE1021: Introduction to Problem solving and programming
VIT Bhopal University

REGISTRATION NO: 25BAI12286

Submitted by: Sharayu Sunil
Date: November 24, 2025

Introduction

The Shopping Cart System is a Python-based command-line application that replicates essential e-commerce shopping cart functionalities. It incorporates principles such as CRUD (Create, Read, Update, Delete) operations, user interaction handling, and transaction management.

The system enables users to browse products, manage cart items, apply discount codes, and complete checkout processes through an intuitive command-line interface.

This project serves as an educational model for understanding user interface design, and practical problem-solving in retail-focused systems.

Problem Statement

Challenge: Users need a way to efficiently manage their shopping experience including:

- Browsing available products with prices
- Adding items to a cart with quantity management
- Modifying removing items from the cart
- Viewing cart contents , also includes subtotals and totals
- Applying discount codes at checkout page
- Processing transactions with automatic calculation of final amounts

Objective: To develop a command-line shopping cart system that allows users to perform complete end-to-end shopping transactions with proper validation.

Functional Requirements

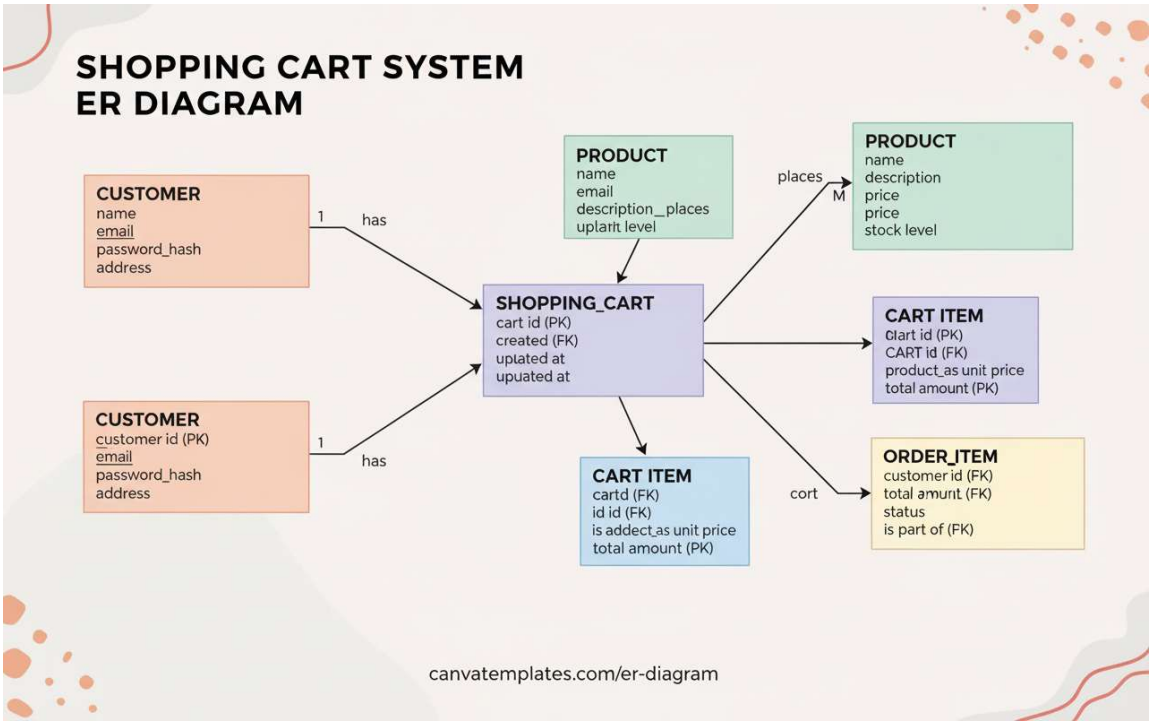
- Display product catalog with ID, name, and price
- Add items to cart with quantity validation
- View cart contents with itemized display
- Update item quantities in the cart
- Delete items from the cart
- Clear entire cart in one action

- Apply discount codes at checkout (FLASH65, STUDENT5, NEWCOMER10)
- Display final amount after discount application
- Handle invalid product IDs
- Provide user-friendly menu-driven interface

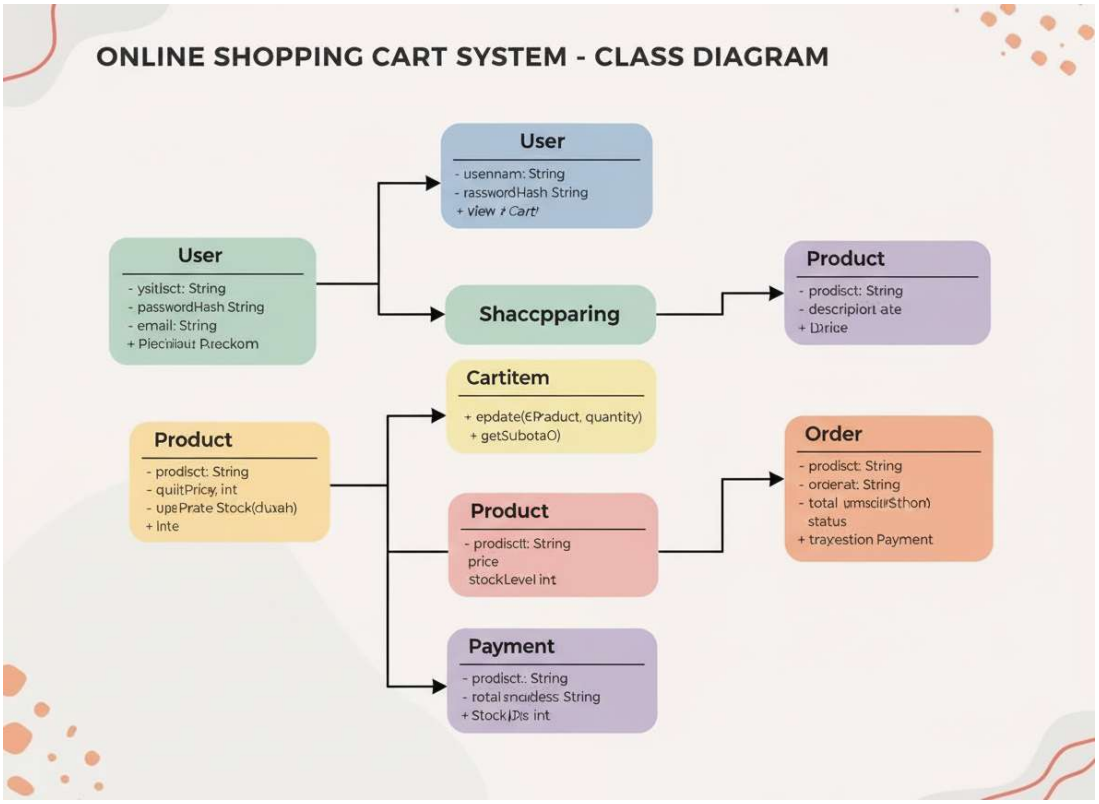
Non-functional Requirements

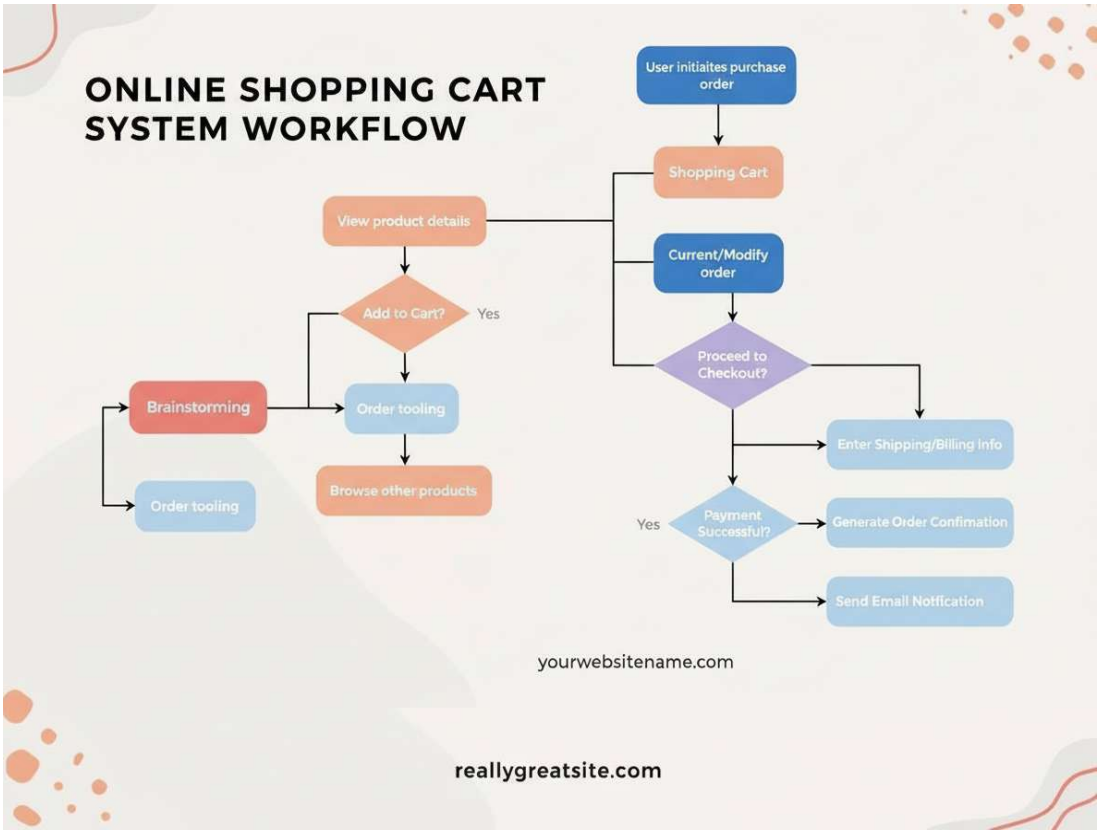
- **Performance:** Cart operations should execute within milliseconds to ensure a smooth user experience.
- **Usability:** The system should provide an intuitive, menu-driven interface with clear prompts and responsive feedback.
- **Reliability:** The application must handle invalid inputs gracefully and prevent crashes.
- **Scalability:** The data structures should allow seamless addition of new products without requiring code modifications.
- **Maintainability:** Code should be clean, well-organized, and use descriptive variable and function names to support future updates.
- **Portability:** The application should run reliably on any platform with Python version 3.6 or higher.
- **Data Validation:** All user inputs must be validated before processing to maintain data integrity and prevent logical errors.

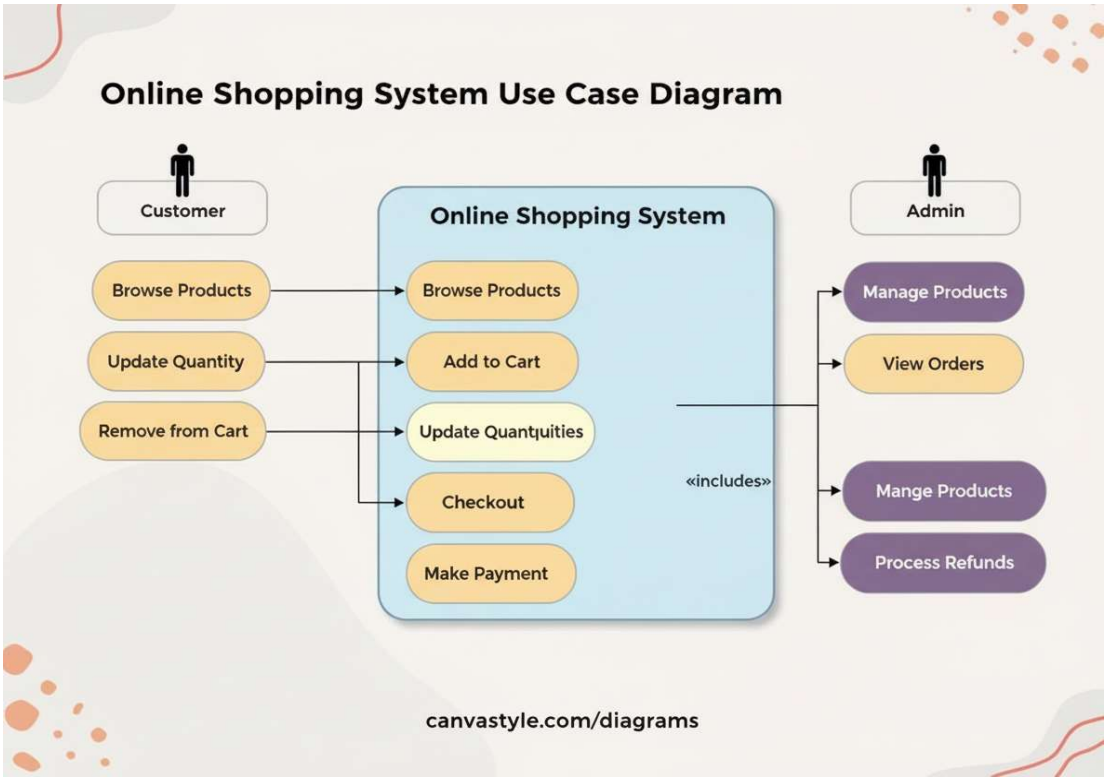
System Architecture

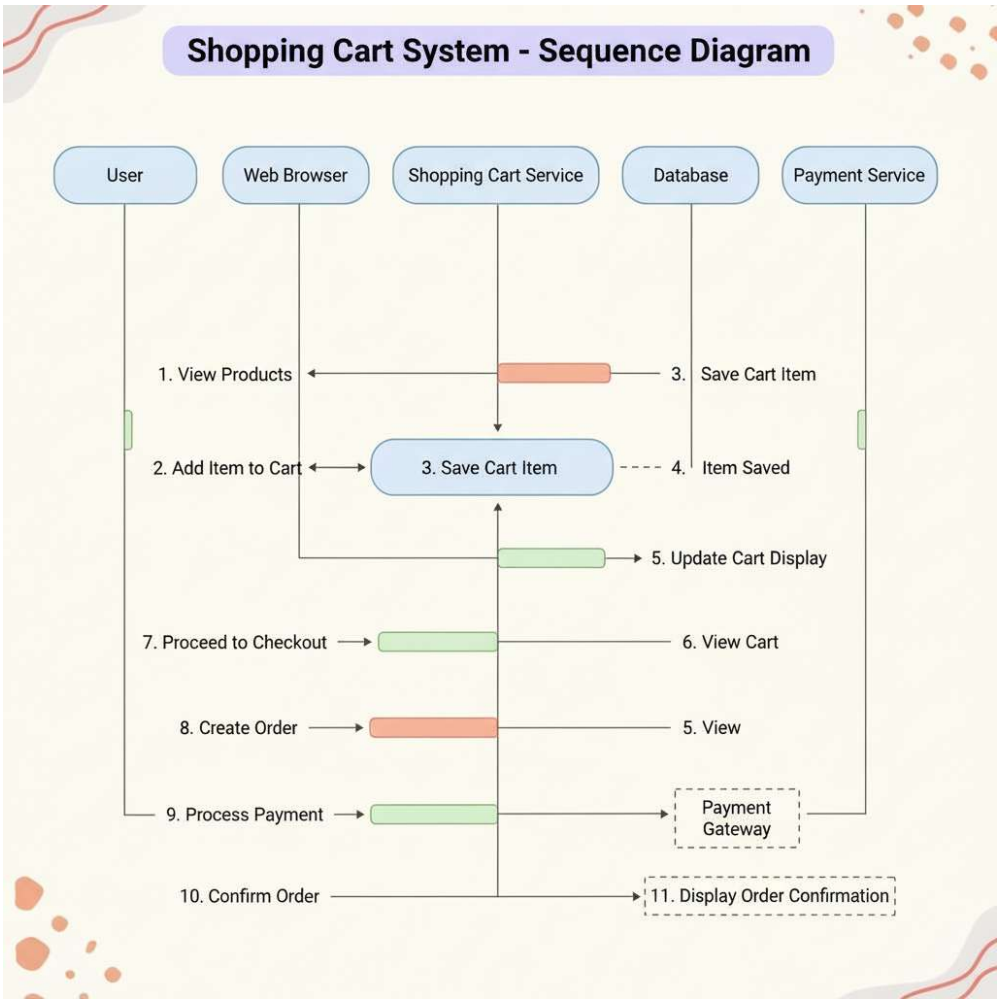


Design Diagrams






Online Shopping System Use Case Diagram
Customer
Online Shopping System
Admin
Browse Products
Update Quantity
Remove from Cart
Browse Products
Add to Cart
Update Quantities
Checkout
Make Payment
«includes»
Manage Products
View Orders
Mange Products
Process Refunds
canvastyle.com/diagrams


Shopping Cart System - Sequence Diagram
User
Web Browser
Shopping Cart Service
Database
Payment Service
1. View Products
3. Save Cart Item
2. Add Item to Cart
3. Save Cart Item
4. Item Saved
5. Update Cart Display
7. Proceed to Checkout
6. View Cart
8. Create Order
5. View
9. Process Payment
Payment Gateway
10. Confirm Order
11. Display Order Confirmation

Design Decisions & Rationale

Design Decision	Rationale	Trade-off
Products as Tuples	Immutable, lightweight, efficient for read-only data	Cannot easily modify product details
Cart as Dict List	Flexible schema, enables easy item modifications	Slightly higher memory than tuples
Menu-Driven Interface	Intuitive for users, clear flow, easy to extend	Limited to CLI (no GUI)
String Discount Codes	User-friendly, easy to remember	Case-sensitive handling needed
Input Validation	Prevents crashes and invalid operations	Extra code, slower input processing
No Database	Simple, no external dependencies	Data lost after program exit
List Comprehension for Search	Pythonic, readable, concise	Scans entire list (O(n) complexity)
Global Data Structures	Easy to manage, direct access	Less modular than OOP approach

Implementation Details

Data Structures

Products List:

```
products = [
(1, "Shirt", 2000),
(2, "Trouser", 3000),
(3, "Cardigan", 1500),
(4, "Skirt", 700),
(5, "Shawl", 400),
(6, "Jeans", 3000),
(7, "Tshirts", 600)
]
```

Cart List :

```
cart = [
{"id": 1, "name": "Shirt", "price": 2000, "quantity": 2},
```

{"id": 3, "name": "Cardigan", "price": 1500, "quantity": 1}
]

Key Functions

find_product(id): Uses generator expression with `next()` to find first product matching ID in O(n) time[1].

cart_index(id): Locates item position in cart for update/delete operations[1].

add_items(id, quantity): Implements add-or-increment logic; creates new cart entry if product not present, otherwise increments quantity[1].

view_cart(): Displays formatted table with itemized costs and running total[1].

update_item(id, quantity): Replaces quantity for existing cart item with validation[1].

delete_item(id): Removes item from cart after existence check[1].

apply_discount(total, code): Implements percentage-based discount with dictionary lookup of discount codes[1].

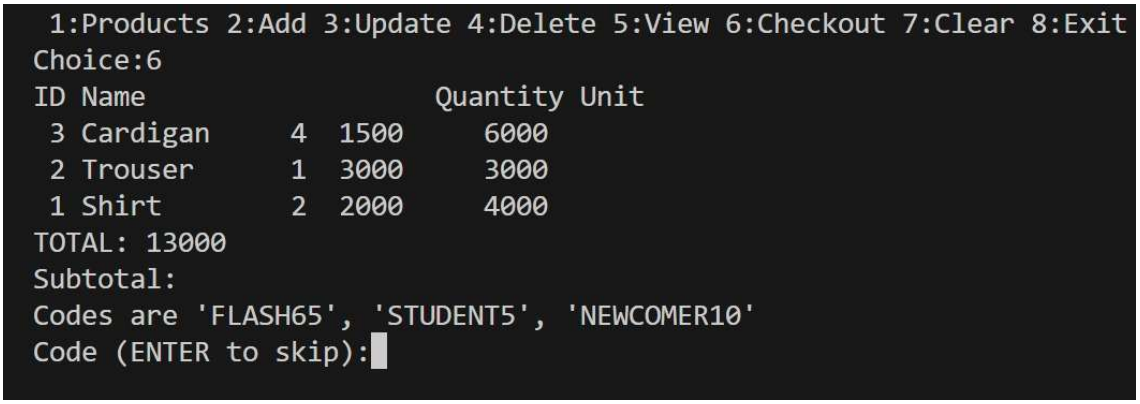
total_amount(): Calculates cart total using generator expression with sum()[1].

---

10. Screenshots / Results

Example Usage Scenario

Step 1: View Products



Step 2: Add Items

```
 1:Products 2:Add 3:Update 4:Delete 5:View 6:Checkout 7:Clear 8:Exit
Choice:1
1,Shirt - Rs 2000
2,Trouser - Rs 3000
3,Cardigan - Rs 1500
4,Skirt - Rs 700
5,Shawl - Rs 400
6,Jeans - Rs 3000
7,Tshirts - Rs 600
 1:Products 2:Add 3:Update 4:Delete 5:View 6:Checkout 7:Clear 8:Exit
Choice:2
id: 3
quantity4
Done
```

Step 3: View Cart

```
 1:Products 2:Add 3:Update 4:Delete 5:View 6:Checkout 7:Clear 8:Exit
Choice:6
ID Name                  Quantity Unit
 3 Cardigan     4  1500     6000
 2 Trouser      1  3000     3000
 1 Shirt        2  2000     4000
TOTAL: 13000
Subtotal:
Codes are 'FLASH65', 'STUDENT5', 'NEWCOMER10'
Code (ENTER to skip):
```

Step 4: Checkout with Discoun

```
 1:Products 2:Add 3:Update 4:Delete 5:View 6:Checkout 7:Clear 8:Exit
Choice:1
1,Shirt - Rs 2000
2,Trouser - Rs 3000
3,Cardigan - Rs 1500
4,Skirt - Rs 700
5,Shawl - Rs 400
6,Jeans - Rs 3000
7,Tshirts - Rs 600
 1:Products 2:Add 3:Update 4:Delete 5:View 6:Checkout 7:Clear 8:Exit
Choice:2
id: 2
quantity1
Done
 1:Products 2:Add 3:Update 4:Delete 5:View 6:Checkout 7:Clear 8:Exit
Choice:6
ID Name                 Quantity Unit
 1 Shirt        9  2000    18000
 3 Cardigan     3  1500     4500
 2 Trouser      4  3000    12000
TOTAL: 34500
Subtotal:
Codes are 'FLASH65', 'STUDENT5', 'NEWCOMER10'
Code (ENTER to skip):FLASH65
Discount: 22425
Final: 12075
Thank You
```

11. Testing Approach

Unit-Level Testing

1. **Product Search:** Test find_product() with valid and invalid IDs
2. **Cart Operations:** Verify add, update, delete with edge cases (negative quantities, duplicate IDs)
3. **Total Calculation:** Confirm accurate math with multiple items
4. **Discount Application:** Validate each discount code (FLASH65, STUDENT5, NEWCOMER10) and invalid codes

Integration Testing

- Test complete workflow: Add → View → Update → Checkout
- Verify data consistency across operations
- Test cart state after clear operation
- Validate menu navigation between options

Edge Cases Tested

- Empty cart checkout (should skip)
- Zero or negative quantities (rejected)
- Non-existent product IDs (handled gracefully)
- Invalid menu choices (error message, loop continues)
- Discount code case insensitivity
- Multiple items with same product ID (quantity incremented)
- Floating-point discount calculations

Challenge 2: Duplicate Products in Cart

Issue: Adding same product multiple times created duplicate cart entries instead of incrementing quantity.

Solution: Modified `add_items()` to check if product already exists using `cart_index()` before appending new entry.

Challenge 3: Discount Code Matching

Issue: User typos in discount codes (case sensitivity, spacing) prevented discount application.

Solution: Normalized input with `.strip().upper()` to make code matching case-insensitive and remove whitespace

.Challenge 4: Menu Loop Control

Issue: Invalid menu choices caused errors or unexpected behavior.

Solution: Added comprehensive if-elif chain with else clause for invalid input, allowing loop to continue with error message.

---

13. Learnings & Key Takeaways

Technical Skills Acquired

- **Python Fundamentals:** Lists, dictionaries, tuples, and their appropriate use cases
- **Functional Programming:** Generator expressions, lambda functions, list comprehensions
- **Error Handling:** Try-except blocks and graceful degradation
- **Input/Output:** Formatted string output and user input validation
- **Algorithm Design:** Linear search, conditional logic, state management
- **Data Structure Selection:** Understanding when to use tuples vs lists vs dictionaries

Software Engineering Principles

- **Modularity:** Breaking complex problems into manageable functions
- **DRY Principle:** Avoiding code duplication through reusable functions
- **User Experience:** Designing intuitive interfaces with clear feedback
- **Robustness:** Anticipating edge cases and handling failures gracefully
- **Maintainability:** Writing readable code with meaningful names and structure

Problem-Solving Insights

- Complex features (checkout, discounts) decompose into simple, testable functions
- Input validation at boundaries prevents cascading errors
- Clear separation of concerns improves debugging and maintenance
- Iterative development catches issues early

---

14. Future Enhancements

Proposed Improvements

1. **Persistent Storage:** Save cart and products to CSV/JSON file for session recovery
2. **Product Categories:** Organize products by clothing type, price range
3. **Search & Filter:** Allow users to search products by name or price range
4. **Inventory Management:** Track stock levels and prevent overselling
5. **User Accounts:** Maintain purchase history and preferences
6. **Graphical Interface:** Convert to web app using Flask/Django or GUI with Tkinter
7. **Payment Integration:** Connect to real payment gateways (Stripe, PayPal)
8. **Advanced Discounts:** Percentage discounts, bulk discounts, seasonal sales
9. **Review System:** Allow customers to rate and review products
10. **Order Tracking:** Generate order IDs and delivery status tracking
11. **Tax Calculation:** Automatically calculate GST based on product category
12. **Mobile App:** Convert to mobile application for iOS/Android

Implementation Priority

Phase 1 (High Priority):

- Persistent data storage
- Inventory management
- Advanced discount system

Phase 2 (Medium Priority):

- Web interface conversion
- User authentication
- Product search/filter

Phase 3 (Low Priority):

- Payment gateway integration
- Mobile application
- Analytics dashboard

References

- https://www.geeksforgeeks.org/python/python-syllabus/
- https://www.geeksforgeeks.org/software-engineering-diagrams-and-models/