Online shopping cart

In

MECHANICAL ENGINEERING

By

SHIVAM RAJ SINGH (Roll No. 2401330400037)
SHREYA JHA (Roll No.2401330400038)
SHUBHAM YADAV (Roll No. 2401330400039)

Under the Supervision of
Mr. Anurag Mishra
Assistant Professor, CS

Mechanical engineering
School of Mechanical Engineering
NOIDA INSTITUTE OF ENGINEERING AND TECHNOLOGY,
GREATER NOIDA
(An Autonomous Institute)
Affiliated to
DR. A.P.J. ABDUL KALAM TECHNICAL UNIVERSITY, LUCKNOW
June, 2025

TABEL OF CONTENT

Certificate

Certified that **SHIVAM RAJ SINGH (Roll No. 2401330400037), SHREYA JHA (Roll No. 2401330400038) SHUBHAM YADAV (Roll No. 2401330400039)** have carried out the project work in Session **2024-25** having "**Shopping cart Management System**" for **B.Tech in MECHANICAL ENGINEERING** from Dr. A.P.J. Abdul Kalam Technical University (AKTU) (formerly UPTU), Lucknow under my supervision. The project report embodies original work, and studies are carried out by the student himself/herself, and the contents of the project report do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date:
Signature:

Mr. Anurag Mishra
Assistant Professor
Department of Computer Science
NIET, Greater Noida

Acknowledgement

I would like to express my sincere gratitude to **NIET(Noida Institute Of Engineering In Technology, Greater Noida)** for allowing me to work on **Shopping Cart Management System**. I am grateful to my Supervisor **Mr. Anurag Mishra Assistant Professor, Computer Science** for their guidance and support throughout the project. I also acknowledge the contributions of Colleagues who helped me complete this project.

Thank you all for your support and encouragement.

Sincerely,

SHIVAM RAJ SINGH

SHREYA JHA

SHUBHAM YADAV

Online Shopping Cart Project Report

Abstract

The **Online Shopping Cart System** is a Python-based console application designed using **Object-Oriented Programming (OOP)** principles. It simulates a basic e-commerce platform where users can **browse physical and digital products**, add them to a cart, modify quantities, and proceed to **checkout**.

The system uses **JSON files** for persistent storage of both the **product catalog** and the **shopping cart**, ensuring data consistency across sessions. Inventory levels are **dynamically updated** in real time as users interact with the cart, enabling accurate stock management.

Key features include:

- Browsing categorized products
- Adding/removing items from the cart
- Updating item quantities
- Displaying a detailed order summary
- Finalizing purchases with inventory adjustment

The OOP structure employs **encapsulation, inheritance, and polymorphism**, enabling clean code organization and reusability. Separate classes handle product definitions, cart operations, and file management.

This project provides a practical understanding of file handling, class design, and interactive user interfaces, making it a useful model for learning **e-commerce application logic** and essential programming concepts.

1. Introduction

This project introduces an interactive **command-line Shopping Cart System** built using **Python** and grounded in **Object-Oriented Programming (OOP)** principles. It simulates a simplified e-commerce experience, allowing users to browse, select, and purchase products through a text-based interface. Products are classified into three types: **base products, physical goods, and digital items**, each with unique properties and behaviours handled through inheritance and polymorphism.

A key feature of the system is **dynamic quantity tracking**, which ensures that inventory levels are updated in real time as users add or remove items from their cart. The cart supports **flexible item management**, including quantity updates, item removal, and a clear summary at checkout.

To ensure continuity between sessions, the system uses **JSON files** to store both the product catalogue and the cart's current state. This allows users to resume their shopping activity seamlessly and ensures data integrity.

The modular and extensible design makes it easy to add new product types or features in the future. By leveraging OOP concepts such as **encapsulation, inheritance, and modularity**, this project provides a clear and maintainable codebase while offering a solid foundation for understanding e-commerce logic in Python.

2. About the Project

This project simulates the backend logic of an e-commerce site. Users can view available products, add/remove/update them in their cart, and proceed to checkout. Inventory is updated in real time. Data persistence ensures continuity across sessions.

Key Features:
- Distinction between Physical and Digital products.
- Modular class design using inheritance and encapsulation.
- Persistent product and cart state using JSON.
- Cart total and subtotal calculations.

2.1 Methodology & Modules

Methodology:
- Requirements gathering for basic cart functionality.
- OOP design with polymorphism for product types.
- Implementation using Python and JSON file storage.
- Manual testing using console-based UI.

Modules:
- Product (base class)
- PhysicalProduct & DigitalProduct (inherit Product)
- CartItem (links product to quantity)
- ShoppingCart (main logic and state control)
- main() loop for user interaction

2.2 Hardware & Software Requirements

Hardware:
- Minimum: Dual-core processor, 2 GB RAM
- Recommended: i3 or higher, 4 GB RAM

Software:
- OS: Windows/Linux/Mac
- Language: Python 3.x
- Libraries: json, os
- Editor: VS Code / PyCharm

Advantages:

- **Reusable Class Structure: The system is built using well-structured OOP classes, allowing components like products, carts, and inventory to be reused or inherited for more complex applications.**
- **Easy Modification and Extension: Thanks to its modular design, new features such as discount handling, payment integration, or user accounts can be added with minimal code changes.**
- **Persistent Session Using JSON: By saving product data and cart state in JSON files, users can resume sessions without losing progress, mimicking real-world e-commerce functionality.**
- **Supports Real-World Product Handling: The differentiation between physical and digital products allows simulation of real scenarios, such as shipping for physical items and instant access for digital goods.**

Applications:

- **Educational OOP Training: This project is ideal for students and beginners to learn and apply core OOP concepts like encapsulation, inheritance, and polymorphism in a practical context.**
- **Prototype Backend for E-commerce: It serves as a foundation for developing more advanced e-commerce systems, especially during the prototyping phase of a web or app-based store.**
- **Inventory and Cart Simulations: Useful for testing how inventory and cart management systems behave, especially in teaching, research, or internal business logic validation scenarios.**

4. Sample Outputs

```
1. View Products:
> Output: ProductID | Name | Price | Stock [+ weight or download
link]

2. Add Item to Cart:
> Product ID: D001
> Quantity: 2
> Output: Item added to cart.

3. View Cart:
> Output: Item: <name>, Qty: <x>, Subtotal: ₹XXX.XX

4. Update Quantity:
> Product ID: D001
> New Quantity: 3
> Output: Quantity updated.

5. Remove Item:
> Product ID: D001
> Output: Item removed.

6. Checkout:
> Output: Grand Total: ₹XXXX.XX
Thank you for shopping.
```

CODE

```
import json
import os
class Product:
    def _init_(self, product_id, name, price, quantity_available):
        self._product_id = product_id
        self._name = name
        self._price = price
        self._quantity_available = quantity_available

    @property
    def product_id(self): return self._product_id

    @property
    def name(self): return self._name

    @property
    def price(self): return self._price

    @property
    def quantity_available(self): return self._quantity_available

    @quantity_available.setter
    def quantity_available(self, value):
        self._quantity_available = max(0, value)

    def decrease_quantity(self, amount):
        if 0 < amount <= self._quantity_available:
            self._quantity_available -= amount
            return True
        return False

    def increase_quantity(self, amount):
        self._quantity_available += amount

    def display_details(self):
        return f"{self.product_id} | {self.name} | ₹{self.price} | Stock:
{self.quantity_available}"

    def to_dict(self):
        return {
            "type": "base",
            "product_id": self._product_id,
            "name": self._name,
            "price": self._price,
            "quantity_available": self._quantity_available
        }


class PhysicalProduct(Product):
    def _init_(self, product_id, name, price, quantity_available, weight):
        super()._init_(product_id, name, price, quantity_available)
        self._weight = weight

    @property
    def weight(self): return self._weight

    def display_details(self):
        return f"{self.product_id} | {self.name} | ₹{self.price} | Stock: {self.quantity_available}
| Weight: {self.weight}kg"
```

```
    def to_dict(self):
        data = super().to_dict()
        data.update({"type": "physical", "weight": self._weight})
        return data



class DigitalProduct(Product):
    def _init_(self, product_id, name, price, quantity_available, download_link):
        super()._init_(product_id, name, price, quantity_available)
        self._download_link = download_link

    @property
    def download_link(self): return self._download_link

    def display_details(self):
        return f"{self.product_id} | {self.name} | ₹{self.price} | Download:
{self.download_link}"

    def to_dict(self):
        data = super().to_dict()
        data.update({"type": "digital", "download_link": self._download_link})
        return data



class CartItem:
    def _init_(self, product, quantity):
        self._product = product
        self._quantity = quantity

    @property
    def product(self): return self._product
    @property
    def quantity(self): return self._quantity

    @quantity.setter
    def quantity(self, value):
        self._quantity = max(0, value)

    def calculate_subtotal(self):
        return self.product.price * self.quantity

    def _str_(self):
        return f"Item: {self.product.name}, Qty: {self.quantity}, Price: ₹{self.product.price},
Subtotal: ₹{self.calculate_subtotal():.2f}"

    def to_dict(self):
        return {"product_id": self.product.product_id, "quantity": self.quantity}


class ShoppingCart:
    def _init_(self, product_catalog_file="products.json", cart_state_file="cart.json"):
        self._product_catalog_file = product_catalog_file
        self._cart_state_file = cart_state_file
        self._catalog = self._load_catalog()
        self._items = {}
        self._load_cart_state()

    def _load_catalog(self):
        if not os.path.exists(self._product_catalog_file):
            return {}
        with open(self._product_catalog_file, "r") as f:
```

```
data = json.load(f)
catalog = {}
for prod in data:
    p_type = prod.get("type")
    if p_type == "physical":
        obj = PhysicalProduct(
            product_id=prod["product_id"],
            name=prod["name"],
            price=prod["price"],
            quantity_available=prod["quantity_available"],
            weight=prod["weight"]
        )
    elif p_type == "digital":
        obj = DigitalProduct(
            product_id=prod["product_id"],
            name=prod["name"],
            price=prod["price"],
            quantity_available=prod["quantity_available"],
            download_link=prod["download_link"]
        )
    else:
        obj = Product(
            product_id=prod["product_id"],
            name=prod["name"],
            price=prod["price"],
            quantity_available=prod["quantity_available"]
        )
    catalog[obj.product_id] = obj
return catalog
```

```
def _load_cart_state(self):
    if not os.path.exists(self._cart_state_file):
        return
    with open(self._cart_state_file, "r") as f:
        data = json.load(f)
        for item in data:
            product_id = item.get("product_id")
            if product_id and product_id in self._catalog:
                prod = self._catalog[product_id]
                self._items[product_id] = CartItem(prod, item["quantity"])

def _save_catalog(self):
    with open(self._product_catalog_file, "w") as f:
        json.dump([prod.to_dict() for prod in self._catalog.values()], f, indent=4)

def _save_cart_state(self):
    with open(self._cart_state_file, "w") as f:
        json.dump([item.to_dict() for item in self._items.values()], f, indent=4)

def add_item(self, product_id, quantity):
    product = self._catalog.get(product_id)
    if product and product.quantity_available >= quantity:
        if product_id in self._items:
            self._items[product_id].quantity += quantity
        else:
            self._items[product_id] = CartItem(product, quantity)
        product.decrease_quantity(quantity)
        self._save_cart_state()
        return True
    return False
```

```
def remove_item(self, product_id):
    if product_id in self._items:
        product = self._catalog[product_id]
        qty = self._items[product_id].quantity
        product.increase_quantity(qty)
        del self._items[product_id]
        self._save_cart_state()
        return True
    return False

def update_quantity(self, product_id, new_quantity):
    if product_id in self._items:
        item = self._items[product_id]
        diff = new_quantity - item.quantity
        if diff > 0 and item.product.quantity_available >= diff:
            item.quantity = new_quantity
            item.product.decrease_quantity(diff)
            self._save_cart_state()
            return True
        elif diff < 0:
            item.quantity = new_quantity
            item.product.increase_quantity(-diff)
            self._save_cart_state()
            return True
    return False

def get_total(self):
    return sum(item.calculate_subtotal() for item in self._items.values())
    def display_cart(self):
        print("\nYour Shopping Cart:")
        if not self._items:
            print("Cart is empty.")
            return
        for item in self._items.values():
            print(item)
        print(f"\nGrand Total: ₹{self.get_total():.2f}")

    def display_products(self):
        print("\nAvailable Products:")
        for product in self._catalog.values():
            print(product.display_details())


def main():
    cart = ShoppingCart()
    while True:
        print("\n===== Online Shopping Cart =====")
        print("1. View Products")
        print("2. Add Item to Cart")
        print("3. View Cart")
        print("4. Update Item Quantity")
        print("5. Remove Item from Cart")
        print("6. Checkout")
        print("7. Exit")
        choice = input("Enter choice (1-7): ")
        if choice == "1":
            cart.display_products()
        elif choice == "2":
```

```
        pid = input("Enter Product ID: ")
        try:
            qty = int(input("Enter Quantity: "))
            if cart.add_item(pid, qty):
                print("Item added to cart.")
            else:
                print("Failed to add item. Check ID or stock.")
        except:
            print("Invalid input.")
    elif choice == "3":
        cart.display_cart()
    elif choice == "4":
        pid = input("Enter Product ID: ")
        try:
            qty = int(input("Enter New Quantity: "))
            if cart.update_quantity(pid, qty):
                print("Quantity updated.")
            else:
                print("Update failed.")
        except:
            print("Invalid input.")
    elif choice == "5":
        pid = input("Enter Product ID: ")
        if cart.remove_item(pid):
            print("Item removed.")
        else:
            print("Item not found.")
    elif choice == "6":
        print(f"Checkout Total: ₹{cart.get_total():.2f}")
        print("Thank you for shopping.")
```

```
            break
        elif choice == "7":
            print("Exiting. Goodbye.")
            break
        else:
            print("Invalid choice.")


if _name_ == "_main_":
    main()
```

OUTPUT

Items available in mart

1. View Products

2. Add Item to Cart

3. View Cart

4. Update Item Quantity

5. Remove Item from Cart

6. Checkout

7. Exit

Enter choice (1-7): 1

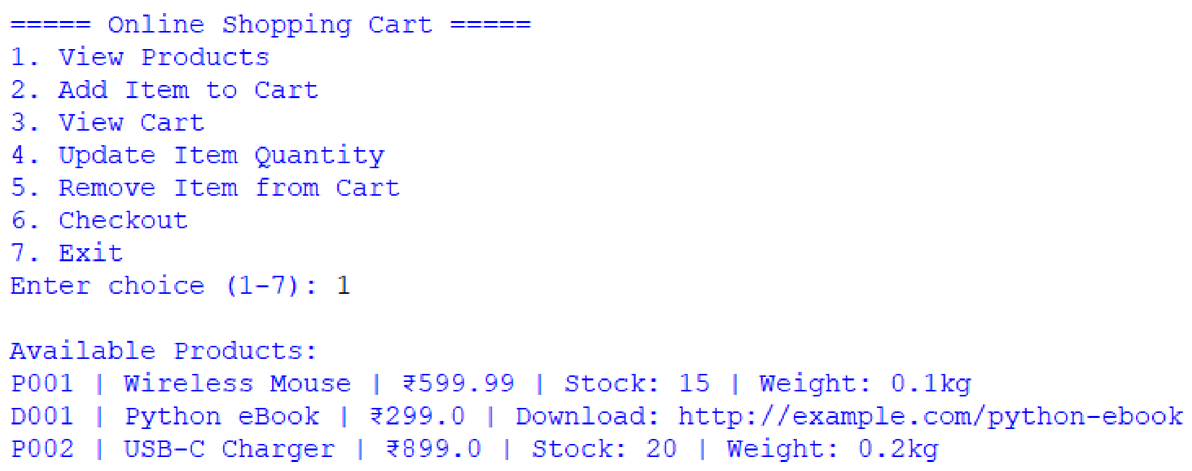
Available Products:

P001 | Wireless Mouse | ₹599.99 | Stock: 15 | Weight: 0.1kg

D001 | Python eBook | ₹299.0 | Download: http://exam

ple.com/python-ebook

P002 | USB-C Charger | ₹899.0 | Stock: 20 | Weight: 0.2kg



```
===== Online Shopping Cart =====
1. View Products
2. Add Item to Cart
3. View Cart
4. Update Item Quantity
5. Remove Item from Cart
6. Checkout
7. Exit
Enter choice (1-7): 1

Available Products:
P001 | Wireless Mouse | ₹599.99 | Stock: 15 | Weight: 0.1kg
D001 | Python eBook | ₹299.0 | Download: http://example.com/python-ebook
P002 | USB-C Charger | ₹899.0 | Stock: 20 | Weight: 0.2kg
```

Items need to be added

1. View Products

2. Add Item to Cart

3. View Cart

4. Update Item Quantity

5. Remove Item from Cart

6. Checkout

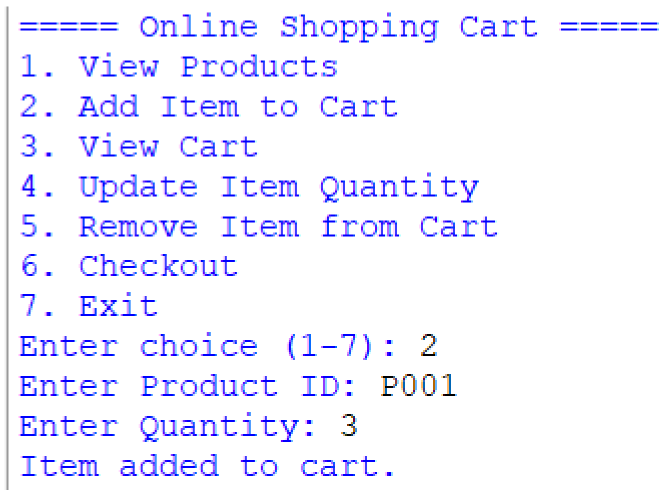
7. Exit

Enter choice (1-7): 2

Enter Product ID: P001

Enter Quantity: 3

Item added to cart.



```
===== Online Shopping Cart =====
1. View Products
2. Add Item to Cart
3. View Cart
4. Update Item Quantity
5. Remove Item from Cart
6. Checkout
7. Exit
Enter choice (1-7): 2
Enter Product ID: P001
Enter Quantity: 3
Item added to cart.
```

To view cart

1. View Products

2. Add Item to Cart

3. View Cart

4. Update Item Quantity

5. Remove Item from Cart

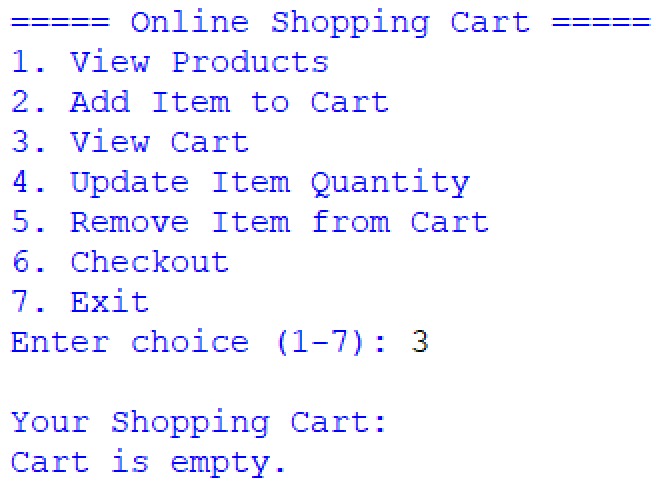
6. Checkout

7. Exit

Enter choice (1-7): 3

Your Shopping Cart:

Cart is empty.



```
===== Online Shopping Cart =====
1. View Products
2. Add Item to Cart
3. View Cart
4. Update Item Quantity
5. Remove Item from Cart
6. Checkout
7. Exit
Enter choice (1-7): 3

Your Shopping Cart:
Cart is empty.
```

To update items

1. View Products

2. Add Item to Cart

3. View Cart

4. Update Item Quantity

5. Remove Item from Cart

6. Checkout

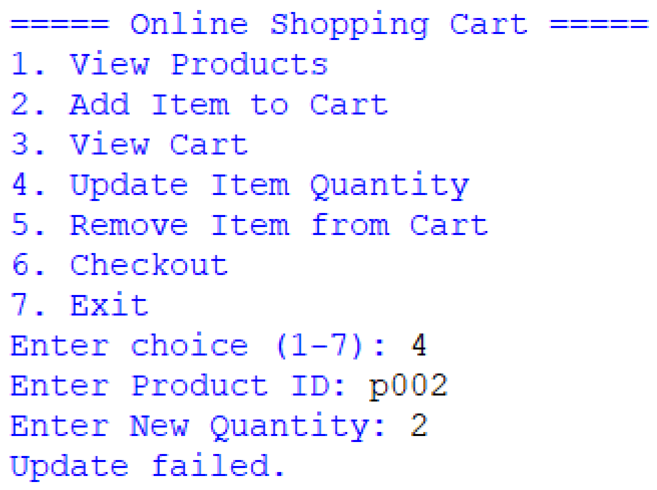
7. Exit

Enter choice (1-7): 4

Enter Product ID: p002

Enter New Quantity: 2

Update failed.



```
===== Online Shopping Cart =====
1. View Products
2. Add Item to Cart
3. View Cart
4. Update Item Quantity
5. Remove Item from Cart
6. Checkout
7. Exit
Enter choice (1-7): 4
Enter Product ID: p002
Enter New Quantity: 2
Update failed.
```

To remove items

1. View Products

2. Add Item to Cart

3. View Cart

4. Update Item Quantity

5. Remove Item from Cart

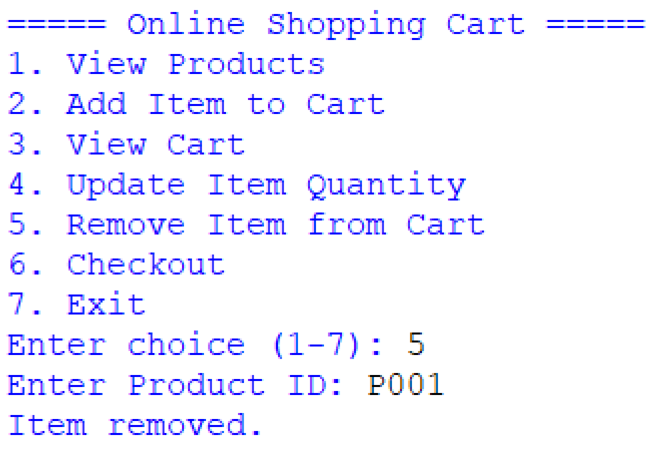
6. Checkout

7. Exit

Enter choice (1-7): 5

Enter Product ID: P001

Item removed.



```
===== Online Shopping Cart =====
1. View Products
2. Add Item to Cart
3. View Cart
4. Update Item Quantity
5. Remove Item from Cart
6. Checkout
7. Exit
Enter choice (1-7): 5
Enter Product ID: P001
Item removed.
```

To Check out

1. View Products

2. Add Item to Cart

3. View Cart

4. Update Item Quantity

5. Remove Item from Cart

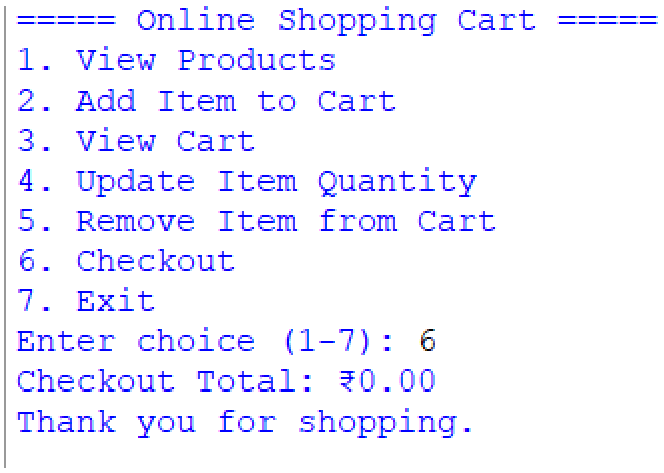
6. Checkout

7. Exit

Enter choice (1-7): 6

Checkout Total: ₹0.00

Thank you for shopping.



```
===== Online Shopping Cart =====
1. View Products
2. Add Item to Cart
3. View Cart
4. Update Item Quantity
5. Remove Item from Cart
6. Checkout
7. Exit
Enter choice (1-7): 6
Checkout Total: ₹0.00
Thank you for shopping.
```

To exit

1. View Products

2. Add Item to Cart

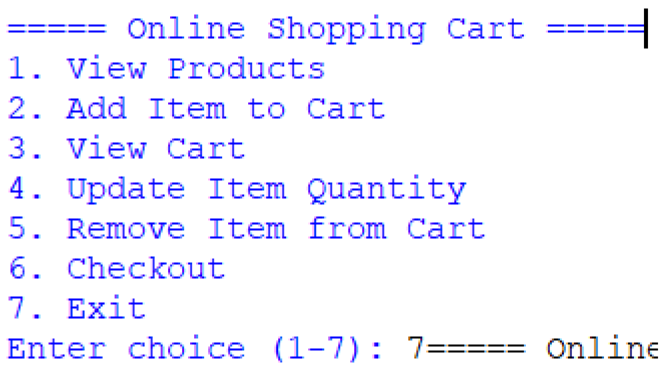
3. View Cart

4. Update Item Quantity

5. Remove Item from Cart

6. Checkout

7. Exit



```
===== Online Shopping Cart =====
1. View Products
2. Add Item to Cart
3. View Cart
4. Update Item Quantity
5. Remove Item from Cart
6. Checkout
7. Exit
Enter choice (1-7): 7===== Online
```

5. References

- **Python Official Documentation**
  https://docs.python.org
  Primary source for Python syntax, built-in functions, file handling, and object-oriented programming.
- **JSON in Python – W3Schools**
  https://www.w3schools.com/python/python_json.asp
  Introductory guide to working with JSON in Python for reading and writing data structures.
- **Object-Oriented Programming in Python – GeeksforGeeks**
  https://www.geeksforgeeks.org/python-oops-concepts
  Detailed explanations of OOP concepts with Python code examples.
- **Real Python – Working with JSON**
  https://realpython.com/python-json
  In-depth tutorial on handling JSON data in Python, including best practices and error handling.
- **Programiz – Python OOP**
  https://www.programiz.com/python-programming/object-oriented-programming
  Beginner-friendly resource for understanding object-oriented design in Python.
- **GitHub – Sample Shopping Cart Projects**
  https://github.com/search?q=python+shopping+cart
  Examples of open-source Python shopping cart projects for reference and comparison.
- **Stack Overflow**
  https://stackoverflow.com
  Community-driven platform for solving development issues and finding best practices related to Python, JSON, and OOP.