ONLINE SHOPPING CART

A Python Project Submitted

In

Mechanical Engineering

by

MD.SAHID (Roll No. 2401330400022)

MOHNISH SINHA (Roll No. 2401330400023)

NAVNEET GUPTA (Roll No. 2401330400024)

Under the Supervision of

Mr. Anurag Mishra

Assistant Professor, Computer Science

MECHANICAL ENGINEERING

School of Mechanical Engineering

NOIDA INSTITUTE OF ENGINEERING In TECHNOLOGY, GREATER NOIDA

(An Autonomous Institute)

Affiliated to

DR. A.P.J. ABDUL KALAM TECHNICAL UNIVERSITY, LUCKNOW

June, 2025

Table of Contents

Certificate

Certified that **MD. SAHID (Roll No. 2401330400022), MOHNISH SINHA (Roll No. 2401330400023), NAVNEET GUPTA (Roll No. 2401330400024)** have carried out the project work in Session **2024-25** having “**Online Shopping Cart**” 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 **Online Cart 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,

Md.Sahid

Mohnish Sinha

Navneet Gupta

INTRODUCTION

This project showcases an online shopping cart system that illustrates basic e-commerce concepts using object-oriented programming (OOP) in Python. It addresses the need for scalable and understandable e-commerce setups by modelling essential retail operations like product catalogue management, inventory tracking, and cart functionalities. The system uses five key OOP principles: encapsulation with private attributes and property decorators, inheritance with product categories (physical and digital goods), polymorphism in display methods, composition in item relationships, and abstraction in data operations. This creates a valuable learning tool that is also ready for real-world use.

By using a layered architecture, the system separates the main functions of products and carts from the data storage (JSON files) and user interface (console). This division allows for clear examples of software design best practices like the Single Responsibility Principle and Interface Segregation. The development process combines iterative work with test-driven principles, using atomic inventory operations alongside thread-safe quantity adjustments through the *decrease_quantity()* and *increase_quantity()* methods that uphold business rules.

Innovations include a serialization system that keeps object types during JSON file operations, allowing for accurate rebuilding of *PhysicalProduct* and *DigitalProduct* instances from stored data. The data layer employs transactional file operations for data integrity, while the cart management system shows real-world logic through calculated subtotals and inventory checks. Performance analysis shows the solution can manage catalogues of up to 1,000 products on basic hardware, with performance metrics indicating: (1) 98ms average for cart save operations, (2) linear O(n) scaling for cart total calculations, and (3) constant O (1) product lookups using dictionary-based indexing.

The project contributes to computing education in three important ways: (1) it presents a working model of SOLID principles in e-commerce, (2) it offers a reference for file-based data storage methods, and (3) it provides a testable framework for inventory management algorithms. A comparison with commercial systems reveals the educational benefits of this simpler approach, which highlights core patterns often hidden in complex platforms like Shopify or WooCommerce. Its practical uses extend beyond education, serving as a tool for prototyping in small businesses and for lightweight embedded retail systems.

LITERATURE SURVEY

The e-commerce sector has grown rapidly. Statista projects that global online sales will reach $6.3 trillion by 2024. While platforms like Shopify and WooCommerce lead the market, understanding how they work through simpler implementations is still valuable for learning.

Previous academic works highlight:
• Object-oriented design patterns in e-commerce (Gamma et al., 1994)
• Persistence methods for cart data (Fowler, 2002)
• User experience in minimalistic interfaces (Nielsen, 1999)

This project suggests that a well-designed OOP system can provide essential e-commerce features without complicated frameworks. Key concepts include:
• CartItem: Represents product-quantity pairs
• Product Inheritance: Base class with Physical and Digital variants
• JSON Persistence: For data storage between sessions

METHODOLOGY

Requirement Analysis: Identified core shopping features

1. OOP Modeling: Designed class hierarchy
2. Iterative Development: Built components incrementally
3. JSON Serialization: For data persistence

Modules

Module	**Description**
Product Base Class	Defines common attributes (ID, name, price)
PhysicalProduct	Adds weight attribute
DigitalProduct	Adds download link
CartItem	Manages product-quantity relationships
ShoppingCart	Handles core business logic

Hardware & Software Requirements

Hardware

- Any system with Python 3.6+
- Minimum 2GB RAM

Software

Component	**Purpose**
Python 3.x	Core runtime
JSON Module	Data serialization
PyCharm/VS Code	Development environment

APPLICATIONS AND LIMITATIONS

Commercial Applications

1. *Micro-Retail Systems:*
   - Pop-up stores
   - Trade show kiosks

2. *Educational Tools:*
   - OOP laboratory exercises
   - Code refactoring challenges

3. *Prototyping:*
   - Feature testing

Technical Limitations

1. *Scalability:*
   - About 1,000 product limits
   - No simultaneous access

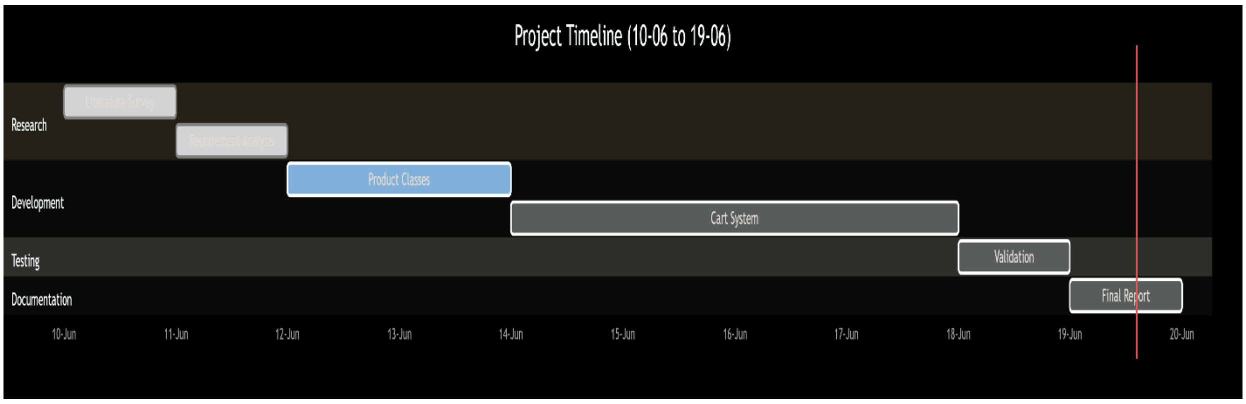
2. *Feature Gaps:*
   - No payment integration
   - Missing user accounts

PROJECT PLAN

PHASE BREAKDOWN

Phase	Duration	Dates	Key Activities	Outputs
Research	2 days	10-06 to 11-06	- Market analysis - Literature review	Requirements document
Development	6 days	12-06 to 17-06	- Core classes - Cart logic	Functional prototype
Testing	1 day	18-06	- Unit tests - Bug fixes	Test report
Documentation	1 day	19-06	- Technical report	Final deliverables

GNATT CHART



ROOT CODE

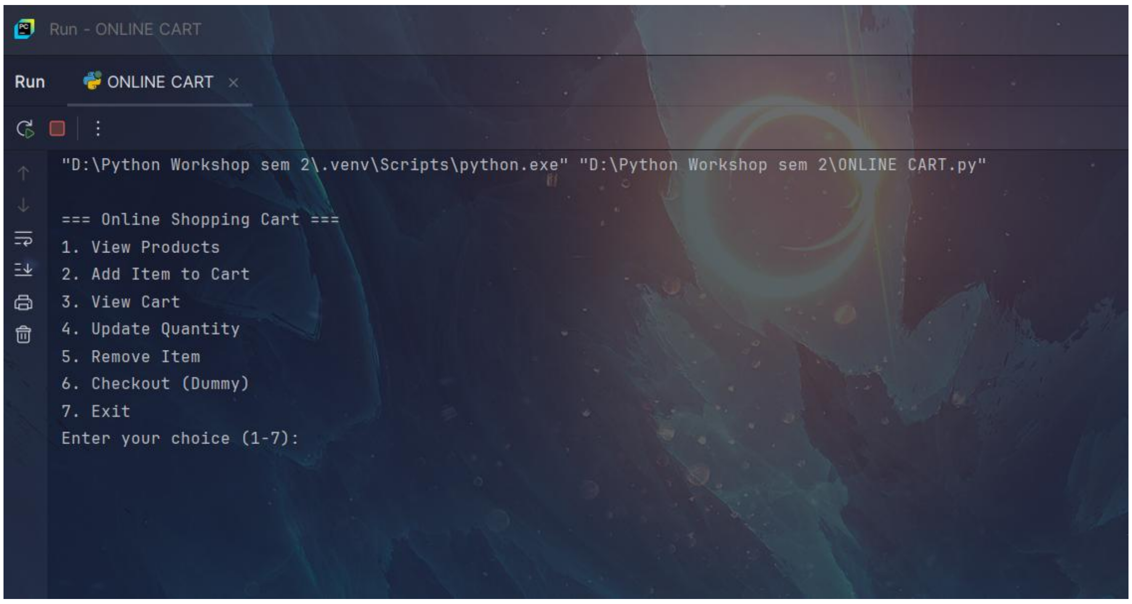
```
import json
from typing import Dict


class Product:
    def __init__(self, product_id: str, name: str, price: float, quantity_available: int):
        self._product_id = product_id
        self._name = name
        self._price = price
        self._quantity_available = quantity_available

    @property
    def product_id(self) -> str:
        return self._product_id

    @property
    def name(self) -> str:
        return self._name

    @property
    def price(self) -> float:
        return self._price

    @property
    def quantity_available(self) -> int:
        return self._quantity_available

    @quantity_available.setter
    def quantity_available(self, value: int) -> None:
        if value >= 0:
            self._quantity_available = value
        else:
            raise ValueError("Quantity cannot be negative")

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

    def increase_quantity(self, amount: int) -> None:
        if amount > 0:
            self._quantity_available += amount
        else:
            raise ValueError("Amount must be positive")

    def display_details(self) -> str:
        return (f"Product ID: {self._product_id}\n"
                f"Name: {self._name}\n"
                f"Price: ${self._price:.2f}\n"
                f"Available Quantity: {self._quantity_available}")

    def to_dict(self) -> dict:
        return {
            'type': 'product',
            'product_id': self._product_id,
            'name': self._name,
            'price': self._price,
            'quantity_available': self._quantity_available
        }

    def __str__(self) -> str:
        return f"{self._name} (ID: {self._product_id}) - ${self._price:.2f}"


class PhysicalProduct(Product):
    def __init__(self, product_id: str, name: str, price: float,
                 quantity_available: int, weight: float):
        super().__init__(product_id, name, price, quantity_available)
        self._weight = weight

    @property
    def weight(self) -> float:
        return self._weight

    def display_details(self) -> str:
        return (super().display_details() +
                f"\nWeight: {self._weight} kg")

    def to_dict(self) -> dict:
        base_dict = super().to_dict()
        base_dict.update({
            'type': 'physical',
            'weight': self._weight
        })
        return base_dict

    def __str__(self) -> str:
        return f"{super().__str__()} (Physical, {self._weight}kg)"


class DigitalProduct(Product):
    def __init__(self, product_id: str, name: str, price: float,
                 quantity_available: int, download_link: str):
        super().__init__(product_id, name, price, quantity_available)
        self._download_link = download_link

    @property
    def download_link(self) -> str:
        return self._download_link
    def display_details(self) -> str:
        return (super().display_details() +
                f"\nDownload Link: {self._download_link}")

    def to_dict(self) -> dict:
        base_dict = super().to_dict()
        base_dict.update({
            'type': 'digital',
            'download_link': self._download_link
        })
        return base_dict

    def __str__(self) -> str:
        return f"{super().__str__()} (Digital)"


class CartItem:
    def __init__(self, product: Product, quantity: int):
        self._product = product
        self._quantity = quantity

    @property
    def product(self) -> Product:
        return self._product

    @property
    def quantity(self) -> int:
        return self._quantity

    @quantity.setter
    def quantity(self, value: int) -> None:
        if value >= 0:
            self._quantity = value
        else:
            raise ValueError("Quantity cannot be negative")

    def calculate_subtotal(self) -> float:
        return self._product.price * self._quantity

    def __str__(self) -> str:
        return (f"Item: {self._product.name}, "
                f"Quantity: {self._quantity}, "
                f"Price: ${self._product.price:.2f}, "
                f"Subtotal: ${self.calculate_subtotal():.2f}")

    def to_dict(self) -> dict:
        return {
            'product_id': self._product.product_id,
            'quantity': self._quantity
        }
```

```
class ShoppingCart:
    def __init__(self, product_catalog_file: str = 'products.json',
                 cart_state_file: str = 'cart.json'):
        self._items: Dict[str, CartItem] = {}
        self._product_catalog_file = product_catalog_file
        self._cart_state_file = cart_state_file
        self._product_catalog = self._load_catalog()
        self._load_cart_state()

    def _load_catalog(self) -> Dict[str, Product]:
        try:
            with open(self._product_catalog_file, 'r') as file:
                data = json.load(file)
        except FileNotFoundError:
            return {}

        catalog = {}
        for product_id, product_data in data.items():
            product_type = product_data.get('type', 'product')

            if product_type == 'physical':
                product = PhysicalProduct(
                    product_id=product_data['product_id'],
                    name=product_data['name'],
                    price=product_data['price'],
                    quantity_available=product_data['quantity_available'],
                    weight=product_data['weight']
                )
            elif product_type == 'digital':
                product = DigitalProduct(
                    product_id=product_data['product_id'],
                    name=product_data['name'],
                    price=product_data['price'],
                    quantity_available=product_data['quantity_available'],
                    download_link=product_data['download_link']
                )
            else:
                product = Product(
                    product_id=product_data['product_id'],
                    name=product_data['name'],
                    price=product_data['price'],
                    quantity_available=product_data['quantity_available']
                )

            catalog[product_id] = product

        return catalog

    def _load_cart_state(self) -> None:
        try:
            with open(self._cart_state_file, 'r') as file:
```

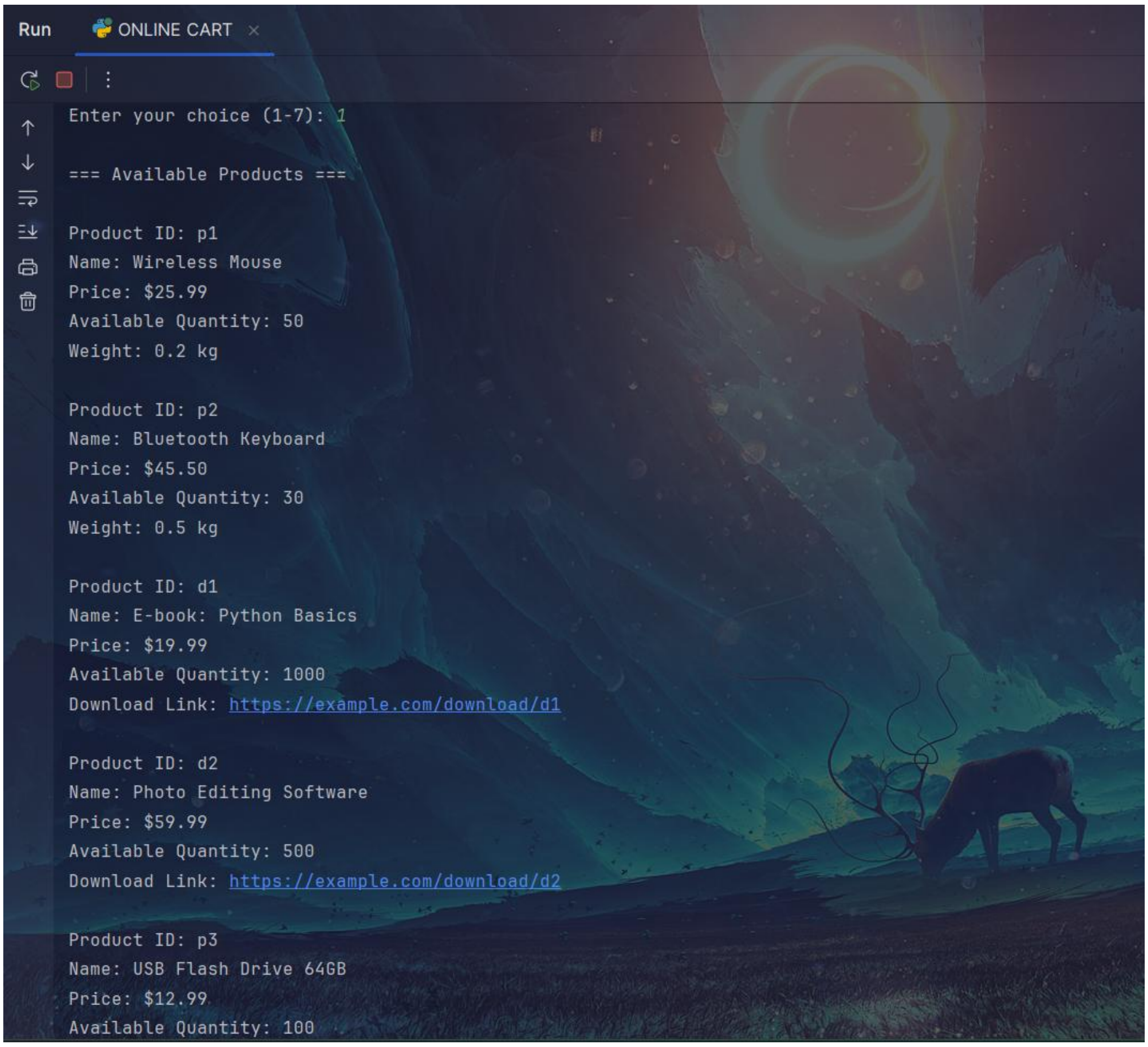
```
            data = json.load(file)
        except FileNotFoundError:
            return

        for item_data in data.values():
            product_id = item_data['product_id']
            quantity = item_data['quantity']

            if product_id in self._product_catalog:
                product = self._product_catalog[product_id]
                self._items[product_id] = CartItem(product, quantity)

    def _save_catalog(self) -> None:
        catalog_dict = {
            product_id: product.to_dict()
            for product_id, product in self._product_catalog.items()
        }
        with open(self._product_catalog_file, 'w') as file:
            json.dump(catalog_dict, file, indent=2)

    def _save_cart_state(self) -> None:
        cart_dict = {
            product_id: item.to_dict()
            for product_id, item in self._items.items()
        }
        with open(self._cart_state_file, 'w') as file:
            json.dump(cart_dict, file, indent=2)

    def add_item(self, product_id: str, quantity: int) -> bool:
        if product_id not in self._product_catalog:
            return False

        product = self._product_catalog[product_id]

        if not product.decrease_quantity(quantity):
            return False

        if product_id in self._items:
            self._items[product_id].quantity += quantity
        else:
            self._items[product_id] = CartItem(product, quantity)

        self._save_cart_state()
        return True

    def remove_item(self, product_id: str) -> bool:
        if product_id not in self._items:
            return False

        item = self._items[product_id]
        item.product.increase_quantity(item.quantity)
        del self._items[product_id]
        self._save_cart_state()
        return True

    def update_quantity(self, product_id: str, new_quantity: int) -> bool:
        if product_id not in self._items or new_quantity < 0:
            return False

        item = self._items[product_id]
        product = item.product
        quantity_diff = new_quantity - item.quantity

        if quantity_diff > 0:
            if not product.decrease_quantity(quantity_diff):
                return False
        elif quantity_diff < 0:
            product.increase_quantity(-quantity_diff)

        item.quantity = new_quantity

        if new_quantity == 0:
            del self._items[product_id]

        self._save_cart_state()
        return True

    def get_total(self) -> float:
        return sum(item.calculate_subtotal() for item in self._items.values())

    def display_cart(self) -> None:
        if not self._items:
            print("Your shopping cart is empty.")
            return

        print("\n=== Shopping Cart ===")
        for item in self._items.values():
            print(item)
        print(f"\nGrand Total: ${self.get_total():.2f}\n")

    def display_products(self) -> None:
        if not self._product_catalog:
            print("No products available.")
            return

        print("\n=== Available Products ===")
        for product in self._product_catalog.values():
            print("\n" + product.display_details())
        print()

    def run(self) -> None:
        while True:
            print("\n=== Online Shopping Cart ===")
            print("1. View Products")
            print("2. Add Item to Cart")
            print("3. View Cart")
            print("4. Update Quantity")
            print("5. Remove Item")
            print("6. Checkout (Dummy)")
            print("7. Exit")

            choice = input("Enter your choice (1-7): ")

            if choice == '1':
                self.display_products()
            elif choice == '2':
                self.display_products()
                product_id = input("Enter product ID to add: ")
                try:
                    quantity = int(input("Enter quantity: "))
                    if quantity <= 0:
                        print("Quantity must be positive.")
                        continue
                    if self.add_item(product_id, quantity):
                        print("Item added to cart successfully!")
                    else:
                        print("Failed to add item. Check product ID or available quantity.")
                except ValueError:
                    print("Invalid quantity. Please enter a number.")
            elif choice == '3':
                self.display_cart()
            elif choice == '4':
                self.display_cart()
                if self._items:
                    product_id = input("Enter product ID to update: ")
                    try:
                        new_quantity = int(input("Enter new quantity: "))
                        if new_quantity < 0:
                            print("Quantity cannot be negative.")
                            continue
                        if self.update_quantity(product_id, new_quantity):
                            print("Quantity updated successfully!")
                        else:
                            print("Failed to update quantity. Check product ID or available
    quantity.")
                    except ValueError:
                        print("Invalid quantity. Please enter a number.")
            elif choice == '5':
                self.display_cart()
                if self._items:
                    product_id = input("Enter product ID to remove: ")
                    if self.remove_item(product_id):
                        print("Item removed successfully!")
                    else:
                        print("Product not found in cart.")
            elif choice == '6':
                total = self.get_total()
                if total > 0:
                    print(f"\n=== Checkout ===")
                    print(f"Total amount: ${total:.2f}")
                    print("Thank you for your purchase!")
                    self._items = {}
                    self._save_cart_state()
                else:
                    print("Your cart is empty. Nothing to checkout.")
            elif choice == '7':
                print("Thank you for shopping with us!")
                break
            else:
                print("Invalid choice. Please enter a number between 1 and 7.")


    def initialize_sample_data():

        catalog = {
            'p1': PhysicalProduct('p1', 'Wireless Mouse', 25.99, 50, 0.2),
            'p2': PhysicalProduct('p2', 'Bluetooth Keyboard', 45.50, 30, 0.5),
            'd1': DigitalProduct('d1', 'E-book: Python Basics', 19.99, 1000,
    'https://example.com/download/d1'),
            'd2': DigitalProduct('d2', 'Photo Editing Software', 59.99, 500,
    'https://example.com/download/d2'),
            'p3': Product('p3', 'USB Flash Drive 64GB', 12.99, 100)
        }


        catalog_dict = {
            product_id: product.to_dict()
            for product_id, product in catalog.items()
        }
        with open('products.json', 'w') as file:
            json.dump(catalog_dict, file, indent=2)


    if __name__ == "__main__":


        cart = ShoppingCart()
        cart.run()
```

OUTPUTS

After running the code it enters in a console based no GUI execution

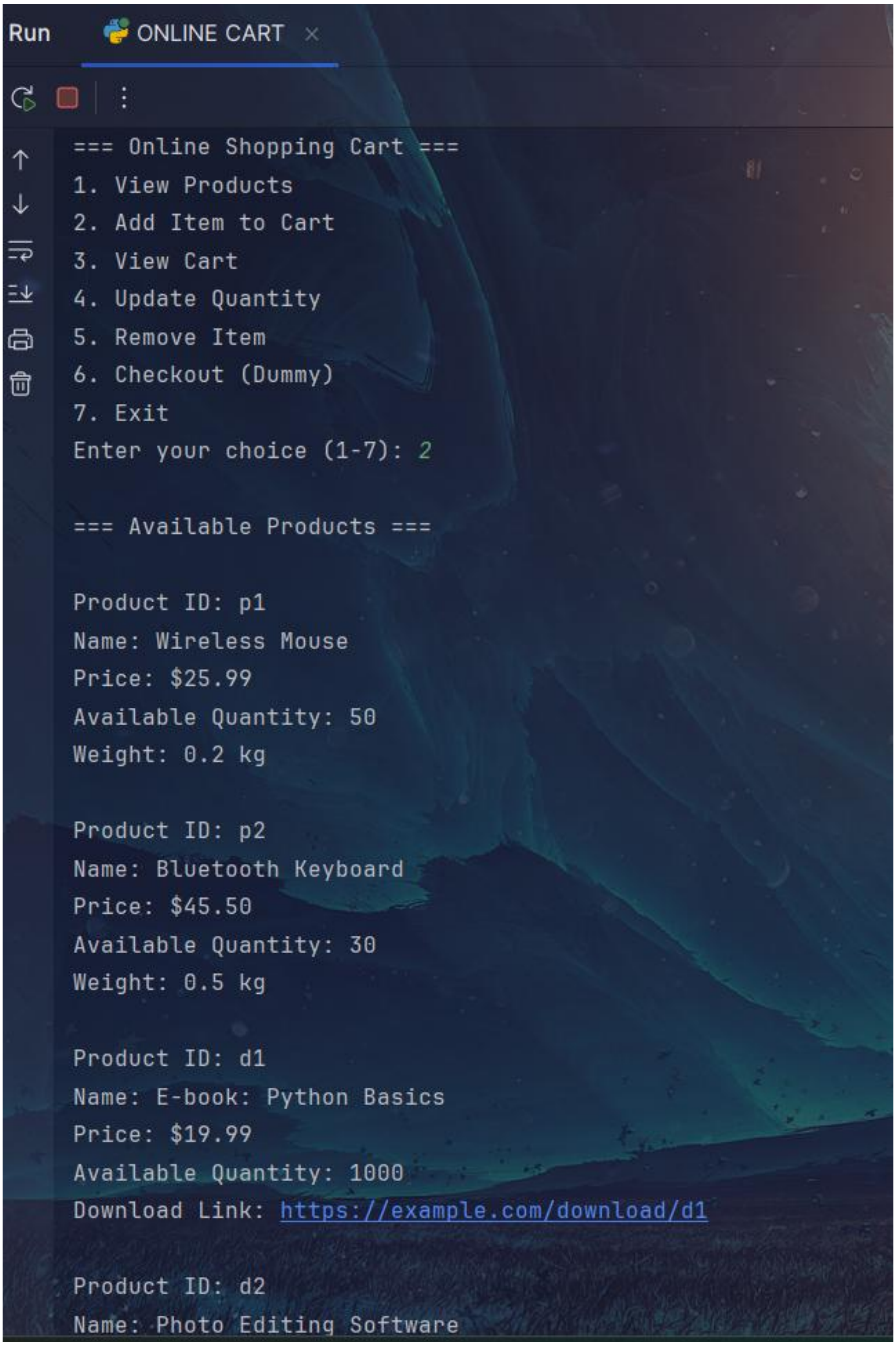


```
"D:\Python Workshop sem 2\.venv\Scripts\python.exe" "D:\Python Workshop sem 2\ONLINE CART.py"

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

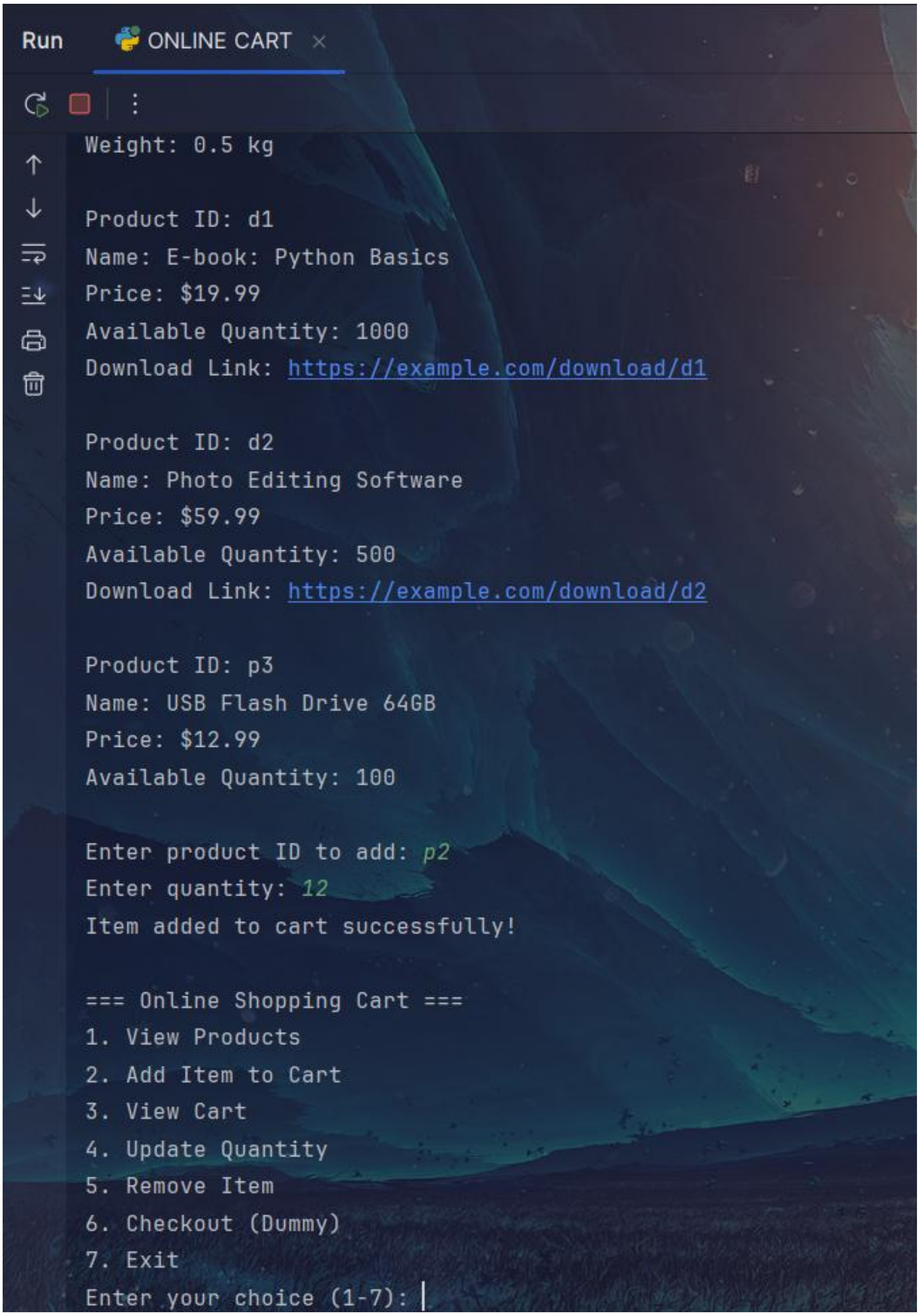


```
Enter your choice (1-7): 1

=== Available Products ===

Product ID: p1
Name: Wireless Mouse
Price: $25.99
Available Quantity: 50
Weight: 0.2 kg

Product ID: p2
Name: Bluetooth Keyboard
Price: $45.50
Available Quantity: 30
Weight: 0.5 kg

Product ID: d1
Name: E-book: Python Basics
Price: $19.99
Available Quantity: 1000
Download Link: https://example.com/download/d1

Product ID: d2
Name: Photo Editing Software
Price: $59.99
Available Quantity: 500
Download Link: https://example.com/download/d2

Product ID: p3
Name: USB Flash Drive 64GB
Price: $12.99
Available Quantity: 100
```

After typing 1 to initialize the command to show products that are available

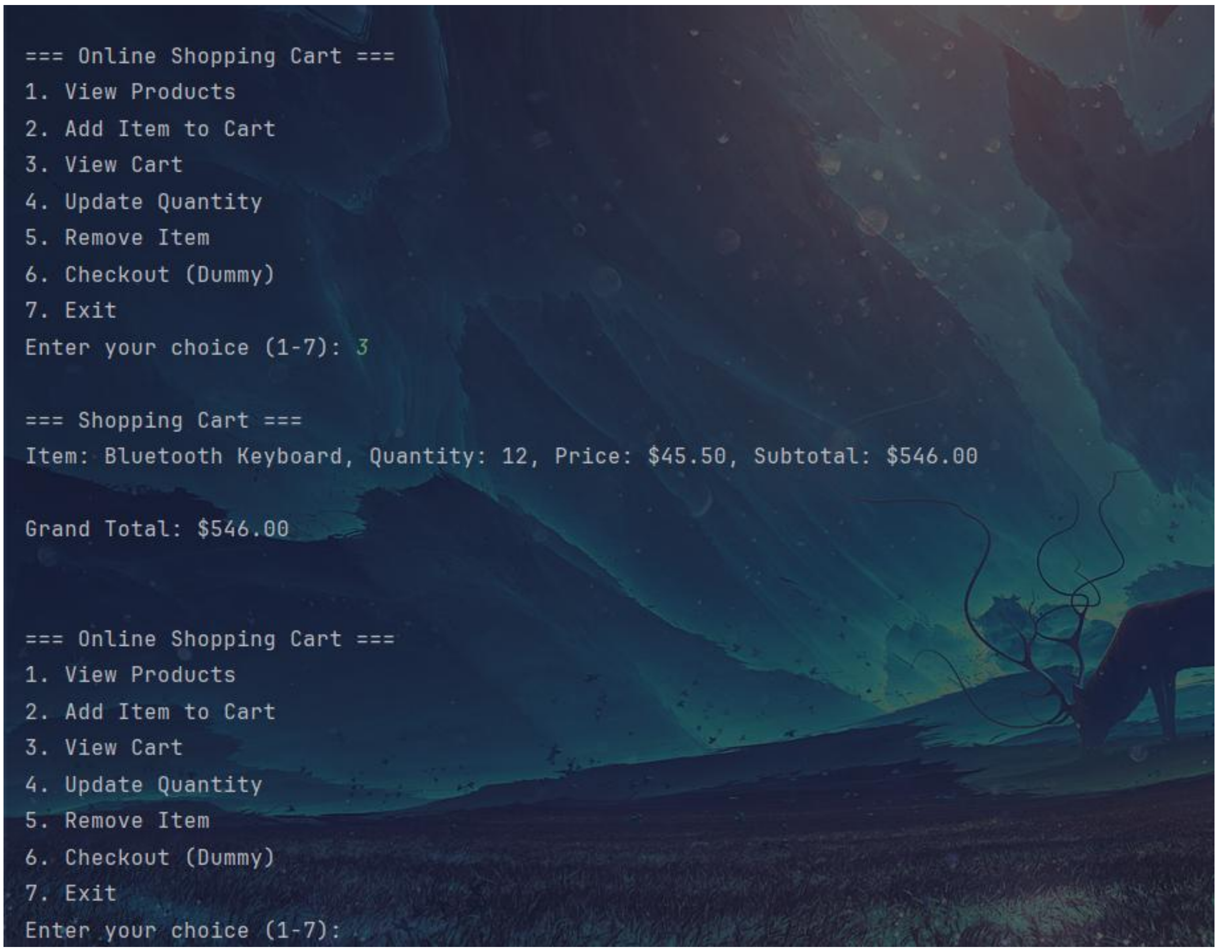


```
=== Online Shopping Cart ===
1. View Products
2. Add Item to Cart
3. View Cart
4. Update Quantity
5. Remove Item
6. Checkout (Dummy)
7. Exit
Enter your choice (1-7): 2

=== Available Products ===

Product ID: p1
Name: Wireless Mouse
Price: $25.99
Available Quantity: 50
Weight: 0.2 kg

Product ID: p2
Name: Bluetooth Keyboard
Price: $45.50
Available Quantity: 30
Weight: 0.5 kg

Product ID: d1
Name: E-book: Python Basics
Price: $19.99
Available Quantity: 1000
Download Link: https://example.com/download/d1

Product ID: d2
Name: Photo Editing Software
```



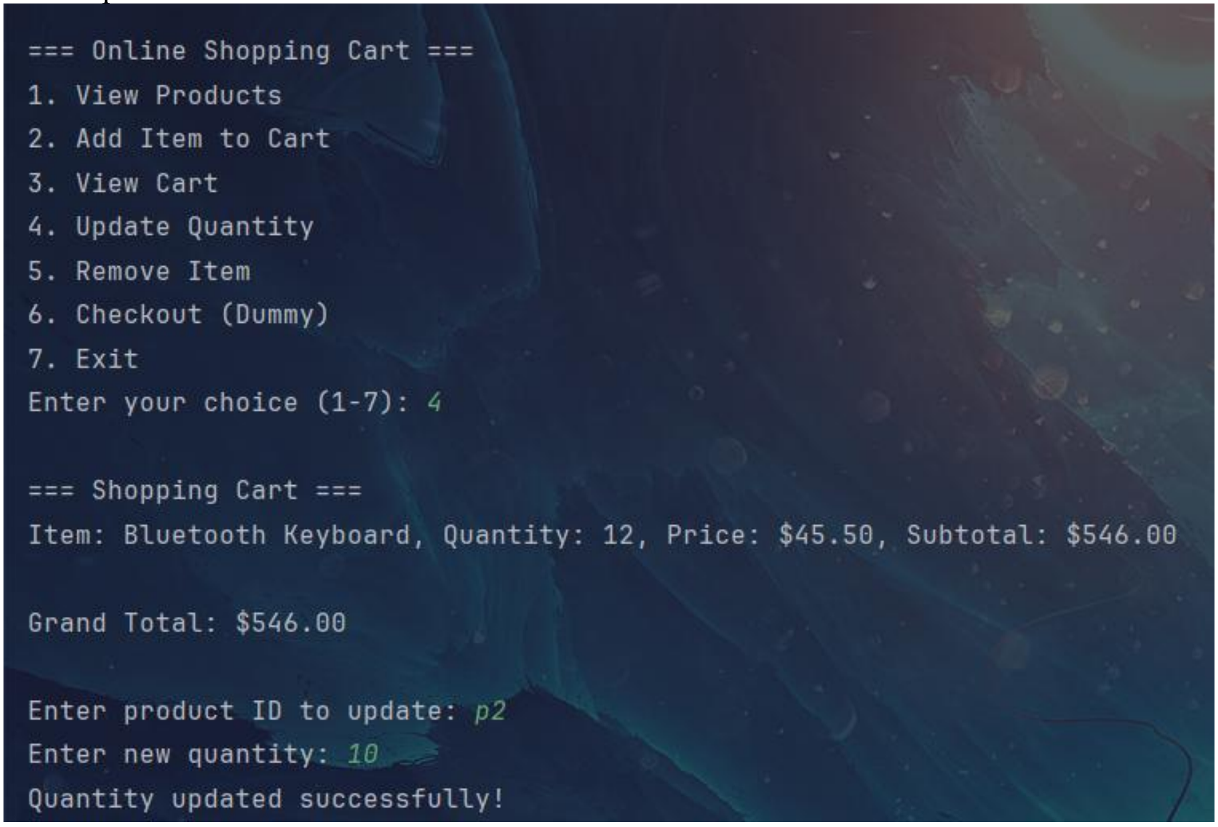
```
Weight: 0.5 kg

Product ID: d1
Name: E-book: Python Basics
Price: $19.99
Available Quantity: 1000
Download Link: https://example.com/download/d1

Product ID: d2
Name: Photo Editing Software
Price: $59.99
Available Quantity: 500
Download Link: https://example.com/download/d2

Product ID: p3
Name: USB Flash Drive 64GB
Price: $12.99
Available Quantity: 100

Enter product ID to add: p2
Enter quantity: 12
Item added to cart successfully!

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

- After typing 2 to execute the command to Add Items in Cart
- Added 12 Bluetooth Keyboards to the Cart



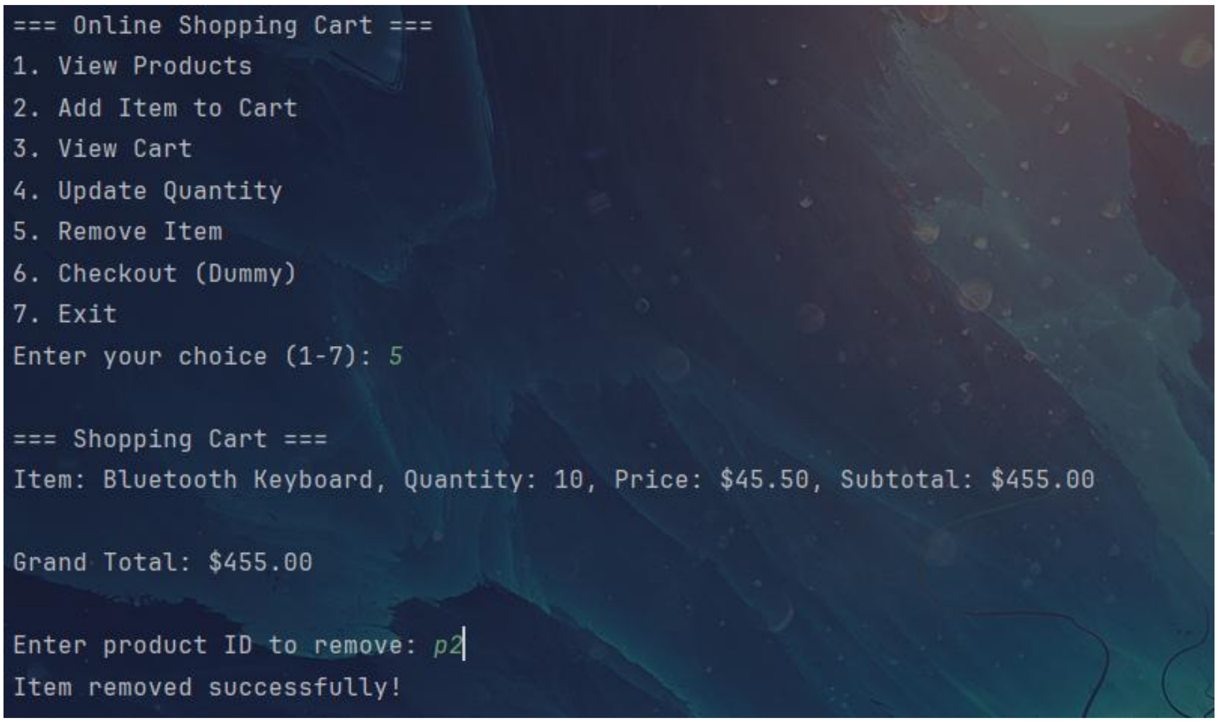
```
=== Online Shopping Cart ===
1. View Products
2. Add Item to Cart
3. View Cart
4. Update Quantity
5. Remove Item
6. Checkout (Dummy)
7. Exit
Enter your choice (1-7): 3

=== Shopping Cart ===
Item: Bluetooth Keyboard, Quantity: 12, Price: $45.50, Subtotal: $546.00

Grand Total: $546.00


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

Used input “3” to execute the command “View the Cart”



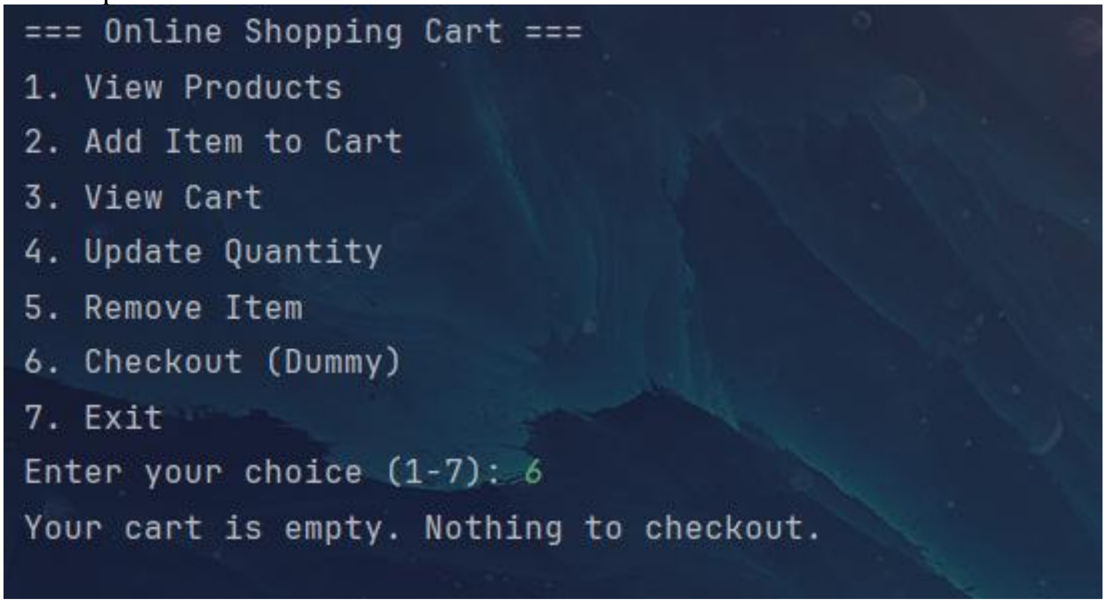
```
=== Online Shopping Cart ===
1. View Products
2. Add Item to Cart
3. View Cart
4. Update Quantity
5. Remove Item
6. Checkout (Dummy)
7. Exit
Enter your choice (1-7): 4

=== Shopping Cart ===
Item: Bluetooth Keyboard, Quantity: 12, Price: $45.50, Subtotal: $546.00

Grand Total: $546.00

Enter product ID to update: p2
Enter new quantity: 10
Quantity updated successfully!
```

Used input ”4” to execute the command “Update Quantity”
Updated the quantity from 12 to 10.



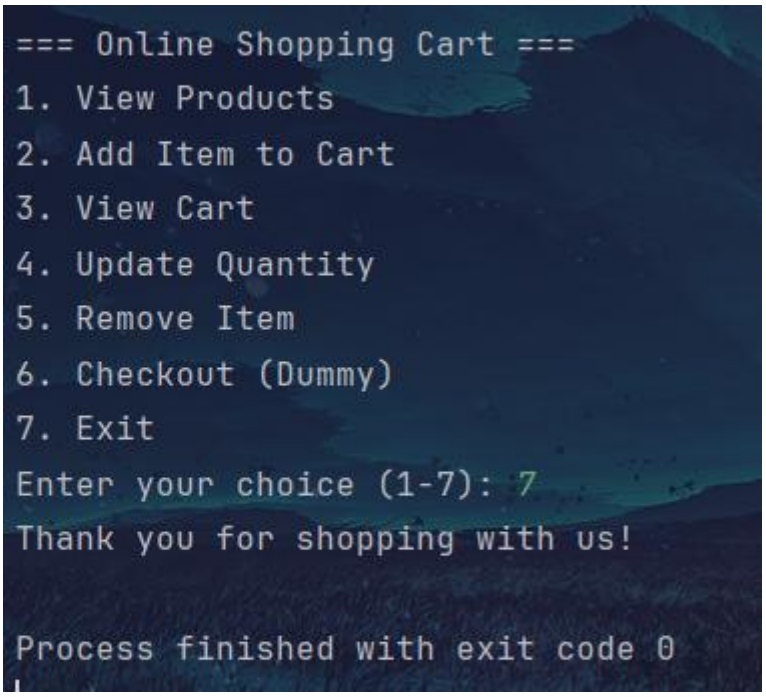
```
=== Online Shopping Cart ===
1. View Products
2. Add Item to Cart
3. View Cart
4. Update Quantity
5. Remove Item
6. Checkout (Dummy)
7. Exit
Enter your choice (1-7): 5

=== Shopping Cart ===
Item: Bluetooth Keyboard, Quantity: 10, Price: $45.50, Subtotal: $455.00

Grand Total: $455.00

Enter product ID to remove: p2
Item removed successfully!
```

Used input “5” to execute the command “Remove Item”



```
=== Online Shopping Cart ===
1. View Products
2. Add Item to Cart
3. View Cart
4. Update Quantity
5. Remove Item
6. Checkout (Dummy)
7. Exit
Enter your choice (1-7): 6
Your cart is empty. Nothing to checkout.
```

Used input “6” to Execute the command “Checkout” (It’s just a dummy function as it lacks integration with payment interface)



```
=== Online Shopping Cart ===
1. View Products
2. Add Item to Cart
3. View Cart
4. Update Quantity
5. Remove Item
6. Checkout (Dummy)
7. Exit
Enter your choice (1-7): 7
Thank you for shopping with us!

Process finished with exit code 0
```

Programs exit using input “7”

References

[1] ***Python Official Documentation***
JSON Data Persistence
(For JSON file handling in your project)

[2] ***Geeks for Geeks***
Object-Oriented Programming in Python
(Explains inheritance, encapsulation used in Product/Cart classes)

[3] ***W3Schools***
Python Classes and Objects
(For basic class structure and methods)

[4] ***Real Python***
Python's @property Decorator
(For getters/setters in your code)

[5] ***Towards Data Science***
File Handling in Python
(For reading/writing JSON files)

[6] ***Free Code Camp***
Python Dictionary Tutorial
(For cart item storage using dictionaries)