Major Project

On

Shopping Cart System

Submitted in The Partial Fulfillment of the Requirement for The Award of The Award of

Diploma in Computer Engineering

(Session 2021-2024)

Submitted By:

Neeraj (211319587888)

Brijesh (211319587885)

Submitted To:

Mrs. HeenaSaini

Department of Computer Engineering **Sri Sukhmani Polytechnic College**

Certificate:

Certified that this project entitled “Shopping Cart System” submitted by Neeraj and Brijesh of Computer Engineering Department, Sri Sukhmani Polytechnic College, Derabassi, as a part of course curriculum of Diploma (Computer Engineering), is a record of student’s own study carried under our supervision and guidance. This report has not been submitted to any other institute for the award of any diploma.

Signature of student   Signature of guide

________________   ________________

____

Signature of HOD

____

2. ACKNOWLEDGMENT

It is great pleasure to present this report on the project "SHOPPING CART SYSYTEM" of diploma curriculum. I am thank you to Sri Sukhmani Polytechnic College for offering such a wonderful challenging opportunity and I express my deepest coordinators for providing all the possible help and their constant encouragement.

I would also like to express our gratitude towards our project guide" Mrs. Heena Sanin placing complete faith and confidence in our ability to carry out this project and providing us his/her time, inspiration, encouragement, help, valuable, guidance Without the sincere and honest guidance of our respected project guide we would have not to reach the present stage.

(NEERAJ)

(BRIJESH)

INDEX

3) ABOUT THE PROJECT

Shopping cart systems are the backbone of online stores. They mimic the experience of a physical shopping cart, allowing customers to browse and add items to a virtual cart. This system calculates totals, taxes, and shipping, and securely processes payments. They are vital for a smooth checkout experience and increasing online sales.

This project is a user-friendly shopping cart system built with Python, utilizing the SQLite database and Tkinter for a graphical user interface (GUI). It offers a convenient and intuitive way for customers to browse products, add them to their cart, and manage their purchases.

Key Features:

- Product Management:
- Add, edit, and delete products from the inventory.
- Maintain product details like name, price, description, and image.
- Cart Functionality:
- Browse products through a user-friendly interface.
- Add items to the cart with quantity adjustments.
- View and manage cart contents, including product details and total cost.
- Easily remove unwanted items.
- Checkout Process:
- Calculate the final bill amount with potential tax inclusions.
- Provides a clear overview of the selected products and their quantities.
- SQLite Database:
- Stores product information and cart data securely.
- Enables persistent data storage, allowing users to revisit their carts later.
- Tkinter GUI:
- Offers a visually appealing and interactive interface for a smooth user experience.

3.1) Introduction:

Shopping Cart System: A Comprehensive Introduction In the realm of e-commerce, the shopping cart system stands as a cornerstone, facilitating a seamless and efficient online shopping experience. This ubiquitous tool empowers customers to browse, select, and accumulate items they wish to purchase, transforming the virtual shopping expedition into a tangible and convenient endeavor. The Essence of the Shopping Cart System At its core, the shopping cart system functions as a virtual repository, akin to a physical shopping cart, where customers can accumulate items they intend to purchase. This temporary storage mechanism streamlines the online shopping process, allowing customers to navigate through the virtual aisles, selecting and adding items to their cart without committing to immediate payment.

These systems allow users to browse products with ease, adding and removing them from their cart as they shop. This flexibility empowers customers to compare items, adjust quantities, and refine their selections before finalizing their purchase. Additionally, features like automatic price calculations and clear overviews of cart contents ensure transparency and informed decision-making.

Essentially, shopping cart systems streamline the online shopping experience, mimicking the familiar process of a physical shopping cart. They provide a convenient and user-friendly way to navigate product selections, manage purchases, and ultimately complete transactions smoothly and efficiently.

Objective:

The objective of this project is to develop a user-friendly shopping cart system that streamlines the online shopping experience. This system will allow customers to browse and add items to their cart, with features like quantity adjustments and item removal. It will calculate running totals, including taxes and shipping costs. Security will be a priority, ensuring safe payment processing. This comprehensive shopping cart system aims to enhance customer convenience and encourage a smooth checkout process, ultimately boosting online sales.

Key Features: User-Friendly Interface: A good shopping cart system prioritizes both customer experience and functionality. Key features include a user-friendly interface for browsing and adding products. Customers should be able to easily view product details, adjust quantities, and remove unwanted items. The system should dynamically calculate subtotal, taxes, and shipping costs.

Dynamic items Handling: The application allows users to add new item update existing ones, and delete outdated information. Through a well-designed system, these operations are performed with simplicity and ease.

Comprehensive Testing: To guarantee the reliability and robustness of the system, a thorough testing phase has been incorporated. This ensures that the shopping cart seamlessly under various scenarios and usage conditions.

Technological Stack: The project is developed using the Python programming language, with the functions , Control statements serves as the underlying Shopping cart system, providing a lightweight yet powerful solution for data storage.

Significance:

this project not only addresses the immediate need for an efficient shopping cart system but also serves as an educational exploration into the realms of GUI development, database design, and software testing. It showcases the practical application of theoretical concepts, making it an insightful journey for both developers and end-users.

As we embark on this project, we aim to deliver a solution that not only meets but exceeds the expectations of users seeking a reliable and user-friendly shopping cart system

4) System Requirements:

1. Hardware Requirements:

• Processor: A processor with a clock speed of 1 GHz or higher is recommended.

• Memory (RAM): A minimum of 2 GB RAM is required for smooth operation. Higher RAM allocations will result in improved performance, especially when handling a large number of contacts.

• Storage: Adequate free storage space to accommodate the application and its associated database. The exact space required will depend on the size of the contact database.

2. Software Requirements:

• Operating System: The application is compatible with Windows, macOS, and Linux operating systems. Ensure that the operating system is up-todate with the latest patches and updates.

• Python: The Contacts Manager is developed using Python programming language. Ensure that a compatible version of Python (3.x) is installed on the system. The application utilizes Python for its logic and functionality.

3.Internet Connection: An internet connection is not mandatory for the standalone operation of the Contacts Manager. However, it may be required for initial installation or updates, depending on the distribution method chosen

4.Display Resolution: A display resolution of 1024x768 pixels or higher is recommended for an optimal viewing experience of the Shopping Cart system GUI. By adhering to these system requirements, users can ensure a smooth and efficient operation of the shopping system GUI App, enhancing their overall experience with contact management

4.2)PRELIIMINARY INVESTIGATION

Preliminary Investigation: Prior to embarking on the development phase, a preliminary investigation was conducted to assess the feasibility and viability of the shopping cart system

GUI App. This phase involved a high-level analysis of the project's requirements, potential challenges, and anticipated benefits.

Key Aspects of Preliminary Investigation: Identification of Stakeholders: Stakeholders were identified, including end-users, developers, and potential contributors.

Understanding the needs and expectations of these stakeholders is crucial for shaping the project's direction.

Market Analysis: An analysis of existing contact management solutions in the market was conducted. This assessment helped identify gaps in functionality and user experience that the shopping cart GUI App seeks to address.

Technical Landscape: The technical landscape was explored, considering the compatibility of the chosen technologies (Python, Tkinter, SQLite) with the targeted user base. This step ensured that the selected tools align with industry standards and trends

4.3)FESABILITY STUDY

• 1.Fesbility Study

This feasibility study assesses the viability of developing a shopping cart system for an ecommerce platform. It analyzes the project's potential across technical, operational, and economic aspects.

1.1Project Description

The proposed shopping cart system aims to provide a user-friendly platform for customers to browse products, add them to their cart, manage quantities, and securely checkout. Key functionalities will include:

- Product browsing and selection
- Adding/removing items from the cart
- Quantity adjustments
- Real-time cart total calculations (including taxes and shipping)
- Secure payment processing integration
- (Optional) User accounts for wish lists, order history, and saved addresses

2.Feasibility Analysis

2.1 Technical Feasibility

- Existing Technologies: Numerous open-source and commercial shopping cart solutions exist, offering a strong foundation for development.
- Development Resources: Development skills in relevant programming languages and frameworks (e.g., PHP, Python, JavaScript) will be required.
- Security Considerations: Secure payment gateways and data encryption will be crucial elements, requiring expertise in security best practices.
- Scalability: The system should be designed to accommodate future growth in product offerings and user base.

2.2 Operational Feasibility

• Target Users: The system should cater to both customers and store owners. • User Interface: An intuitive and user-friendly interface is essential for a smooth shopping experience.

• Integration: The system should integrate seamlessly with existing website infrastructure and potential inventory management systems.

• Maintenance: Ongoing maintenance and updates will be required to ensure system performance and security.

2.3 Economic Feasibility

• Development Costs: The cost of development will depend on chosen technology, internal resources, or potential outsourcing.

• Maintenance Costs: Ongoing maintenance and potential future upgrades need to be factored in. 13

• Return on Investment (ROI): Increased sales and customer satisfaction can lead to a positive ROI.

• Market Analysis: Researching competitor offerings and market trends will help determine the system's competitive advantage.

3. Conclusion

Based on the initial assessment, developing a shopping cart system appears feasible. Existing technologies and skilled developers can overcome any technical challenges. User-centric design and system integration ensure operational success. A thorough costbenefit analysis will determine the project's economic viability.

5)SYSTEM DESIGN

System Design:

The System Design phase involves conceptualizing the overall structure and architecture of the Shopping cart System. This section outlines the key components and their interactions, ensuring a robust foundation for the application's functionality.

Components of System Design:

Architecture: The system follows a client-server architecture where the client represents the cart information with the user, In this way user and system interact with each other .

Modularity: The Shopping cart is divided into modular components, each responsible for specific functionalities such as user interface handling, database operations, and logic. This modular design enhances maintainability and ease of future updates.

User Interface Design: oriented programing s utilized for cart system providing a responsive and visually appealing interface. The design adheres to usability principles, offering an intuitive layout for seamless navigation and interaction.

Error Handling: Robust error-handling mechanisms are integrated to enhance the application's resilience. Users receive informative messages in case of input errors, ensuring a smooth and user-friendly experience.

5.1) Input Design:

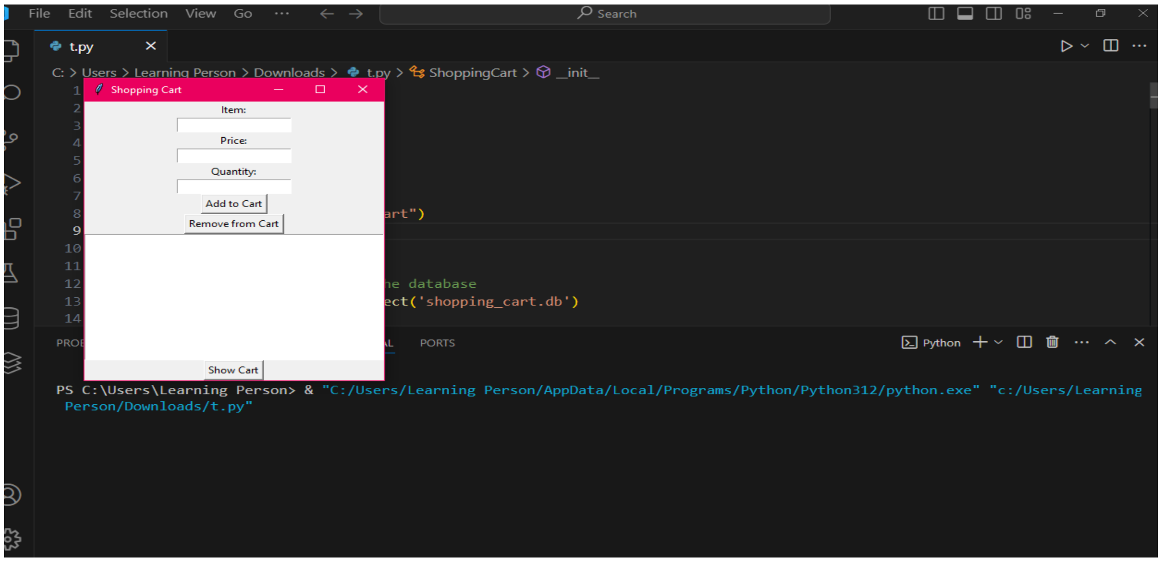
The Input Design phase focuses on defining the methods and processes for collecting user input. It aims to ensure that the user can input data accurately and efficiently into the shopping cart system

The input introduction system within a shopping cart system plays a crucial role in seamlessly integrating user interaction with the GUI interface. This system is designed to provide a user-friendly and intuitive experience for adding, removing, and managing items within the virtual cart.

Strategies for Input Design:

Data Entry Forms:

The application features well-designed data entry forms for adding and updating items Each items makes bill in the end reducing the likelihood of errors.



```
art")
he database
ect('shopping_cart.db')
```

```
PS C:\Users\Learning Person> & "C:/Users/Learning Person/AppData/Local/Programs/Python/Python312/python.exe" "c:/Users/Learning
 Person/Downloads/t.py"
```

Validation Checks: Input fields incorporate validation checks to ensure the correctness of entered data. For example, age is validated to accept only numeric values, preventing data inconsistency .

User-Friendly Input: The design prioritizes user-friendliness, minimizing the number of mandatory fields and providing clear instructions. Default values and placeholders are strategically used to guide users and expedite data entry.

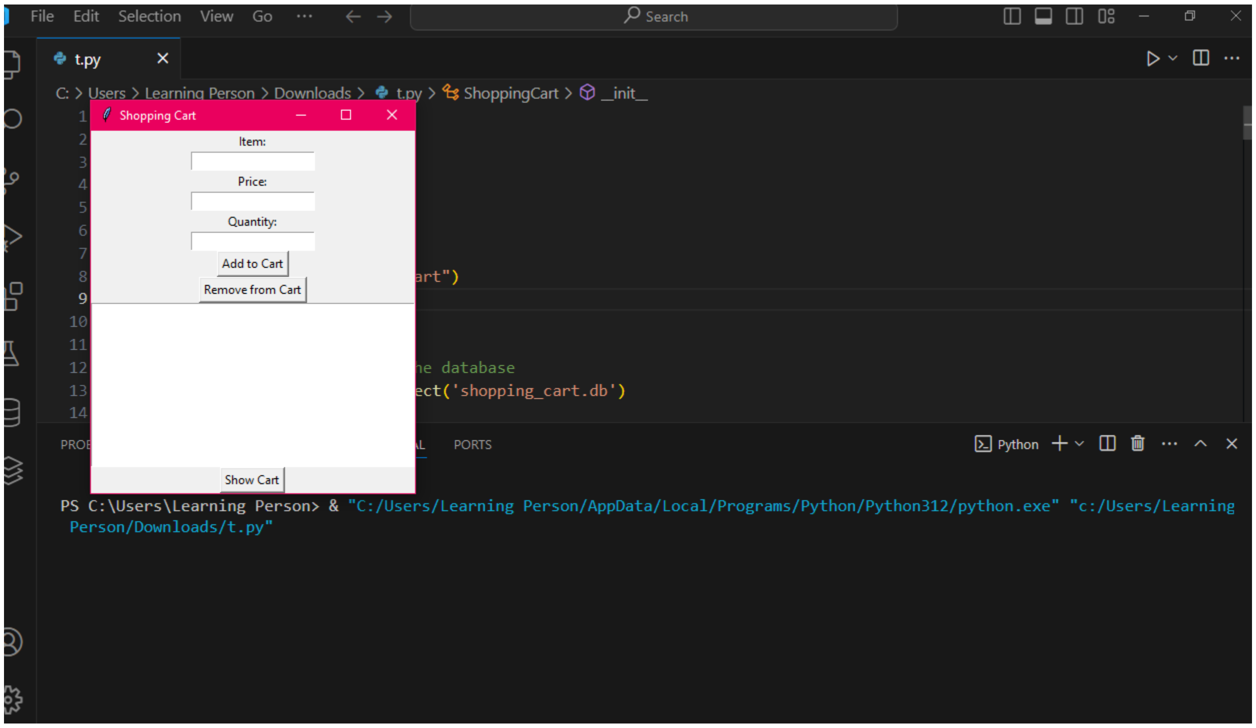
Dropdown Menus and Radio Buttons: Dropdown menus and radio buttons are employed for selecting gender, enhancing the user experience by offering predefined options and reducing the chance of input errors

5.2)OUTPUT DESIGN

Output Design: The Output Design phase outlines how information is presented to users, ensuring clarity and accessibility. This includes the presentation of data on the user interface, as well as any reports or notifications generated by the system..

Elements of Output Design:

User Interface Display: The system displays information in a simple form , providing an organized view. About items, quantity and in the items total bill This presentation facilitates easy comprehension and navigation


File Edit Selection View Go
Search
t.py
C: > Users > Learning Person > Downloads > t.py > ShoppingCart > __init__
Shopping Cart
Item:
Price:
Quantity:
Add to Cart
Remove from Cart
Show Cart
PORTS
Python

```
art")
he database
ect('shopping_cart.db')
```

```
PS C:\Users\Learning Person> & "C:/Users/Learning Person/AppData/Local/Programs/Python/Python312/python.exe" "c:/Users/Learning
 Person/Downloads/t.py"
```

Feedback Messages:

The system generates feedback messages to inform users about the success or failure of operations. These messages are displayed in a visually distinct manner, ensuring users are promptly informed about the outcome of their actions.

Error Messages:

In case of errors, detailed error messages are provided, guiding users on corrective actions. These messages are designed to be informative and user-friendly, aiding users in resolving issues efficiently.

Printable Reports:

The shopping cart system offers the functionality to generate printable reports of biils of items details. These reports are formatted for easy readability and can be used for offline reference or sharing

6)Database Design:

add_product: Adds a Product object to the products list.

display_products: Prints a formatted list of available products with their IDs, names, and prices.

add_to_cart: Checks if the product exists, adds the product and quantity to the cart list, and displays a message.

calculate_total: Calculates the total cost by summing the product prices multiplied by their respective quantities.

generate_bill: Prints a detailed bill with itemized details, including quantity, product name, and individual item price.

7) Testing Phases

Unit Testing: Test individual components of the shopping cart system in isolation. Ensure that each function and method behaves as expected. Verify calculations, validations, and error handling within each component.

Integration Testing: Test the interaction between different modules of the shopping cart system. Verify that components communicate and integrate correctly. Test data flow between modules, such as adding items to the cart, updating quantities, and removing items.

System Testing: Test the entire shopping cart system as a whole. Verify end-to-end functionality from browsing products to completing a purchase. Test various scenarios such as adding multiple items, applying discounts, and selecting different shipping options.

User Acceptance Testing (UAT):Involve end users or stakeholders to validate the system meets their requirements. Test usability, accessibility, and user interface design.Gather feedback to identify any issues or improvements needed before deployment.

Performance Testing: Measure the performance of the shopping cart system under different load conditions.Test response times for adding items, updating quantities, and completing transactions.Identify any bottlenecks or performance issues and optimize system performance accordingly.

Security Testing: Test the security of the shopping cart system to ensure the protection of sensitive customer data. Check for vulnerabilities such as SQL injection, cross-site scripting (XSS), and authentication flaws.Verify encryption of data transmission and storage to prevent unauthorized access.

Regression Testing: Re-test previously validated functionalities after making changes or updates to the system. Ensure that new features or fixes do not introduce new defects or break existing functionality. Use automated regression tests to expedite the testing process and maintain product quality over time.

Compatibility Testing: Test the shopping cart system across different browsers, devices, and operating systems. Ensure consistent performance and user experience across various platforms.Verify compatibility with different screen sizes, resolutions, and input methods.

Localization Testing: Test the shopping cart system with different languages, currencies, and regional settings. Verify that translations are accurate and culturally appropriate. Test date formats, currency symbols, and address formats to ensure compliance with local

8)Conclusion and Future Scope:

The development of the shopping cart marks a significant milestone, providing users with an efficient and user-friendly platform for managing their items purchased information. As we conclude this phase of the project, several key observations and considerations come to the forefront.

Future scope: Shopping cart system successfully addressed the initial objectives set forth during its conceptualization. Key achievements include:

Intuitive User Interface: The system boasts a user-friendly interface developed using satements, facilitating seamless interaction for users of varying technical backgrounds.

Reliable Data Management: We can expand data liberay as much as we can , structured following normalization principles, ensures secure and efficient storage of items details.

Comprehensive Testing: Rigorous testing, spanning unit testing, integration testing, functional testing, and more, has been conducted to identify and rectify potential defects. This has contributed to the overall reliability and stability of the system

Responsive Design: system has been designed to be responsive, catering to various screen resolutions and ensuring an optimal user experience.

Security Measures: Security testing has been conducted to identify and address potential vulnerabilities, enhancing the robustness of the application against common security threats

Weak Areas and Refactoring: The code can be optimized by using a more efficient data structure for storing products, such as a dictionary or a list of dictionaries, instead of the current list of Product objects. This could improve performance, especially for handling a large number of products.

Future Scope :

- Enhanced Personalization:
- Recommendation Engine: Integrate a recommendation engine to suggest products based on user browsing history, purchase history, and similar customer behavior.
- Smart Shopping Lists: Allow users to create personalized shopping lists based on past purchases, dietary preferences, or upcoming occasions.
- Targeted Promotions: Implement targeted promotions and discounts based on user demographics and purchase history.
- Advanced Product Information and Visualization:
- 360° Product Views: Enable users to view products in 360 degrees for a more immersive shopping experience.
- Augmented Reality (AR): Integrate AR features that allow users to virtually place products in their homes to visualize scale and fit.
- Rich Product Content: Include detailed product descriptions, high-resolution images, and user-generated reviews to enhance product information.
- Streamlined Checkout Process:
- Guest Checkout: Offer a guest checkout option for faster purchases without requiring account creation.
- Multiple Payment Options: Integrate various payment gateways to offer users a wider range of payment choices.
- One-Click Ordering: Implement one-click ordering for faster checkouts for returning customers with saved information.
- Integration with External Services:
- Loyalty Programs: Integrate with loyalty program APIs to allow users to earn and redeem points during checkout.
- Inventory Management Systems: Integrate with real-time inventory management systems to ensure product availability and prevent overselling.
- Shipping and Fulfillment Services: Offer real-time shipping quotes and integrate with fulfillment services for efficient order processing.
- Smart Search: Implement AI-powered search functionalities that understand user intent and offer more relevant product suggestions.

9): BIBLIOGRAPHY

Python Software Foundation. Python Language Reference, version 3.9.6. https://docs.python.org/3/reference/

The official documentation for Python provided valuable insights into the language's syntax, features, and best practices.

9.1) BOOKS

• Python for Everyone Cay S. Horstmann 2019

• Head First Python Paul Barry 2009

• Py OOP Dusty Phillips 2020

• Introduction to Computer Science and Data Structures Using Python James V. Mooney

9.2)Links :

/www.geeksforgeeks.org

/https://www.javatpoint.com/python-tutorial .tutorialpoint.com
http://www.w3schools.com

tack Overflow Community. (Various dates). https://stackoverflow.com/

The Stack Overflow community was an invaluable resource for troubleshooting, problemsolving, and gaining insights into best practices.

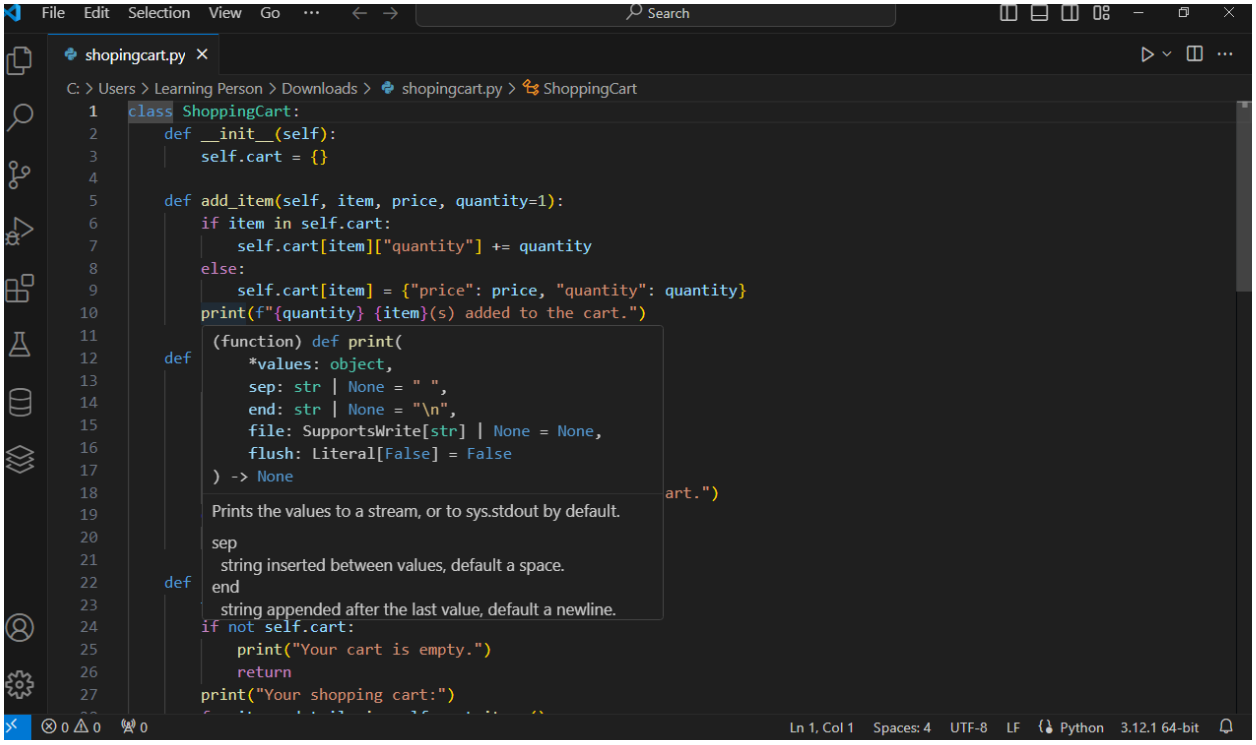
GitHub. (Various repositories).

https://github.com/

Various open-source repositories on GitHub provided code snippets, inspiration, and solutions to specific programming challenges

10): APPENDIX

#DEF



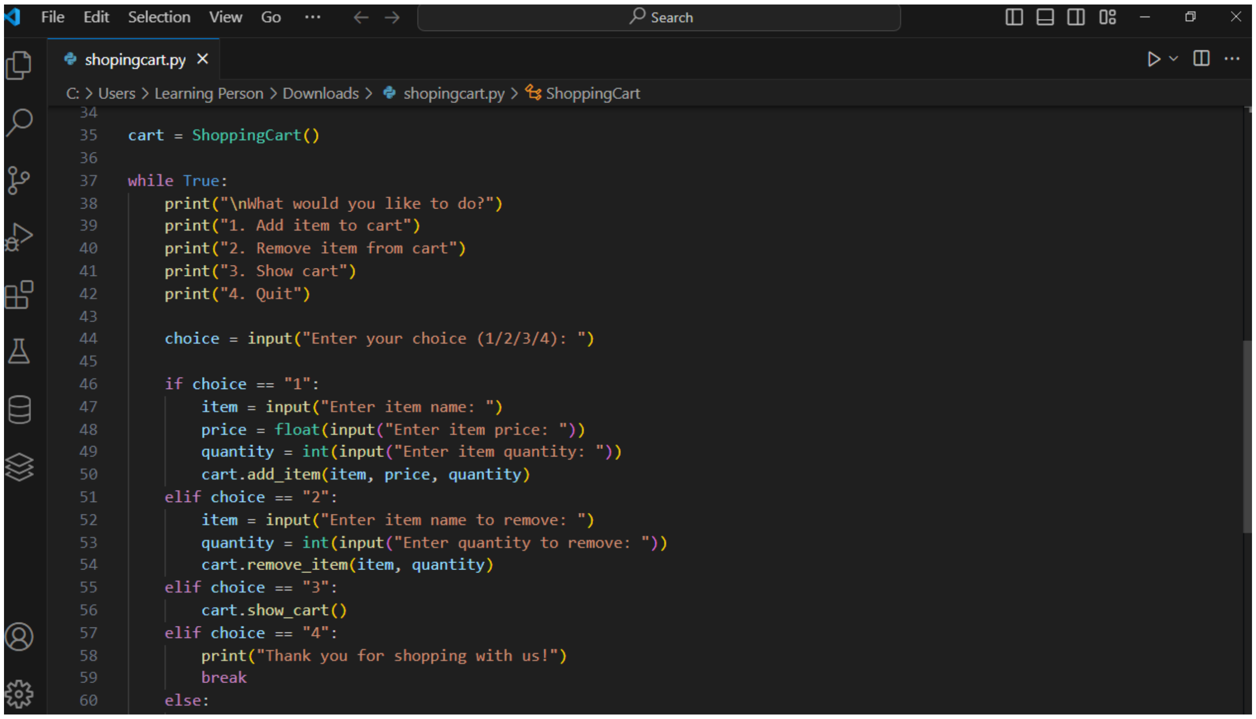
```
class ShoppingCart:
    def __init__(self):
        self.cart = {}

    def add_item(self, item, price, quantity=1):
        if item in self.cart:
            self.cart[item]["quantity"] += quantity
        else:
            self.cart[item] = {"price": price, "quantity": quantity}
        print(f"{quantity} {item}(s) added to the cart.")

    def

                                                      art.")



    def

        if not self.cart:
            print("Your cart is empty.")
            return
        print("Your shopping cart:")
```

```
(function) def print(
    *values: object,
    sep: str | None = " ",
    end: str | None = "\n",
    file: SupportsWrite[str] | None = None,
    flush: Literal[False] = False
) -> None
```

Prints the values to a stream, or to sys.stdout by default.

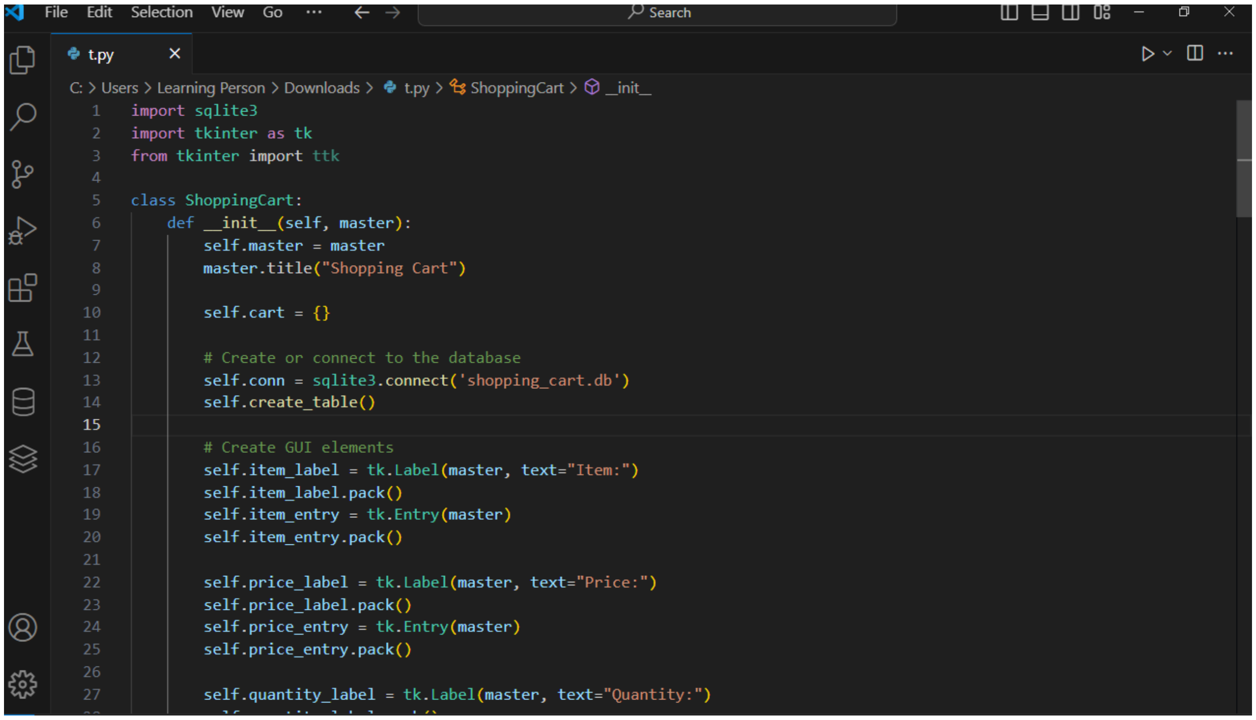
sep
 string inserted between values, default a space.
end
 string appended after the last value, default a newline.

#WHILE LOOP


File Edit Selection View Go ···
Search
shopingcart.py
C: > Users > Learning Person > Downloads > shopingcart.py > ShoppingCart

```python

cart = ShoppingCart()

while True:
    print("\nWhat would you like to do?")
    print("1. Add item to cart")
    print("2. Remove item from cart")
    print("3. Show cart")
    print("4. Quit")

    choice = input("Enter your choice (1/2/3/4): ")

    if choice == "1":
        item = input("Enter item name: ")
        price = float(input("Enter item price: "))
        quantity = int(input("Enter item quantity: "))
        cart.add_item(item, price, quantity)
    elif choice == "2":
        item = input("Enter item name to remove: ")
        quantity = int(input("Enter quantity to remove: "))
        cart.remove_item(item, quantity)
    elif choice == "3":
        cart.show_cart()
    elif choice == "4":
        print("Thank you for shopping with us!")
        break
    else:
```

#GUI



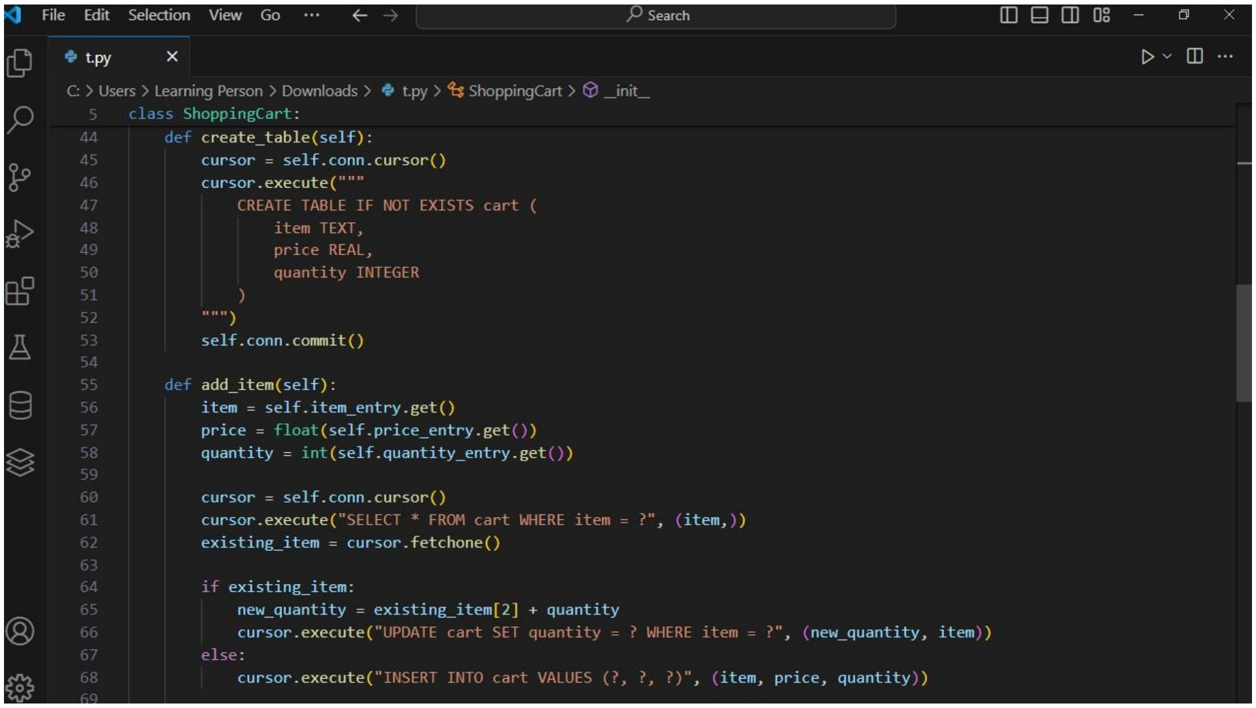
File Edit Selection View Go ··· Search

t.py

C: > Users > Learning Person > Downloads > t.py > ShoppingCart > __init__

```python
import sqlite3
import tkinter as tk
from tkinter import ttk

class ShoppingCart:
    def __init__(self, master):
        self.master = master
        master.title("Shopping Cart")

        self.cart = {}

        # Create or connect to the database
        self.conn = sqlite3.connect('shopping_cart.db')
        self.create_table()

        # Create GUI elements
        self.item_label = tk.Label(master, text="Item:")
        self.item_label.pack()
        self.item_entry = tk.Entry(master)
        self.item_entry.pack()

        self.price_label = tk.Label(master, text="Price:")
        self.price_label.pack()
        self.price_entry = tk.Entry(master)
        self.price_entry.pack()

        self.quantity_label = tk.Label(master, text="Quantity:")
```

#SQLITE FUNCTION



File Edit Selection View Go ···
Search
t.py
C: > Users > Learning Person > Downloads > t.py > ShoppingCart > __init__

```
class ShoppingCart:
    def create_table(self):
        cursor = self.conn.cursor()
        cursor.execute("""
            CREATE TABLE IF NOT EXISTS cart (
                item TEXT,
                price REAL,
                quantity INTEGER
            )
        """)
        self.conn.commit()

    def add_item(self):
        item = self.item_entry.get()
        price = float(self.price_entry.get())
        quantity = int(self.quantity_entry.get())

        cursor = self.conn.cursor()
        cursor.execute("SELECT * FROM cart WHERE item = ?", (item,))
        existing_item = cursor.fetchone()

        if existing_item:
            new_quantity = existing_item[2] + quantity
            cursor.execute("UPDATE cart SET quantity = ? WHERE item = ?", (new_quantity, item))
        else:
            cursor.execute("INSERT INTO cart VALUES (?, ?, ?)", (item, price, quantity))
```