**General Overview**

This app provides a straightforward and user-friendly interface for selecting and ordering food from multiple restaurants, with core functionalities centered around menu selection, cart management, and bill calculation.  
  
  
**Login System:**

* It defines functions for sign-up, login, and a selector for the main functionalities.
* The users dictionary stores usernames and passwords.
* Login validates username and password before proceeding.

**Food Ordering:**

* It has functions for displaying restaurants, searching by name or food item, adding items to the cart, viewing the cart, and checking out.
* It includes price calculation for food items.

**Medicine Ordering:**

* It provides a dictionary with medicine names, prices, and descriptions.
* Users can order medicine and view information about them.

**Groceries:**

* It allows users to view available items, add them to the cart, view the cart, and checkout.

**Customer Support:**

* It offers options for chat with AI or a human assistant.
* The AI can answer general queries, medicine availability, order-related issues, and payment/refunds.

**FOOD DELIVERY**

The app appears to be a food ordering application where users can select food items from various restaurants and add them to a cart. The app allows users to view the menu from different restaurants, add multiple items to their cart, and calculate the total bill based on their selections.

**Functionalities:**

1. **Display Restaurants:** The app presents a list of available restaurants for the user to choose from.

2. **Display Menu:** Once a restaurant is selected, the app displays the menu of that restaurant.

3. **Add Items to Cart:** Users can add multiple food items from the selected restaurant to their cart.

4. **Calculate Total Bill:** The app calculates the total bill for the items added to the cart, based on predefined prices.

5. **Handle Errors:** Basic error handling is implemented for scenarios where items may not exist in the predefined price list.

**Functions:**

1. **add\_to\_car(restaurants)**

- **Purpose:** Manages the process of selecting a restaurant and adding food items to the cart.

- **Functionality:**

- Displays a list of available restaurants.

- Prompts the user to choose a restaurant.

- Displays the menu of the chosen restaurant.

- Calls the `cart` function to add items to the cart.

- Displays the total bill after items have been added.

- **Usage:** This function is the entry point for the user to interact with the app's core functionalities.

2. **cart(foods, foodcart)**

- **Purpose:** Allows users to add multiple items to their cart from the selected restaurant's menu.

- **Functionality:**

- Continuously prompts the user to enter the number of the food item they wish to add to the cart.

- Adds the selected item to the cart.

- Stops adding items when the user enters `0`.

- **Usage:** This function handles the item selection process and updates the cart with the chosen items.

3. **calculate\_bill(foodcart)**

- **Purpose:** Calculates the total cost of the items in the cart.

- **Functionality:**

- Iterates over the items in the cart.

- Adds up the prices of the items based on a predefined price list.

- Returns the total bill amount.

- Handles cases where an item might not be found in the price list.

- **Usage:** This function provides the total cost of the user's selections, which is crucial for finalizing the order.

4. **food()**

- **Purpose:** Placeholder function, presumably used to handle errors or redirect the user to the main menu.

- **Functionality:**

- Prints a message indicating a return to the main menu.

- **Usage:** This function is called when an error occurs (e.g., an item is not found in the price list).

**Additional Components:**

- **prices Dictionary:**

- Contains the prices of various food items.

- Used by the **calculate\_bill** function to determine the cost of items in the cart.

- **restaurants Dictionary:**

- Contains the menus of different restaurants.

- Used by the **add\_to\_car** function to display the menu based on the user's restaurant selection.

**Example Workflow:**

1. **User selects a restaurant:**

- The app displays a list of restaurants.

- User inputs their choice.

- The app displays the menu of the selected restaurant.

2. **User adds items to the cart:**

- The app repeatedly prompts the user to add items from the menu.

- User selects items, which are added to the cart.

- User can stop adding items by entering `0`.

3. **Total bill calculation:**

- The app calculates the total cost of the items in the cart.

- Displays the total bill to the user.

**GROCERY ORDERING**

The Simple Grocery Ordering System is a basic command-line application designed to simulate an online grocery shopping experience. It allows users to view available grocery items, add items to their cart, view their cart, and proceed to checkout. The application is structured with a user-friendly menu interface, guiding users through the shopping process step by step.

**Functionalities**

1. **View Available Items**
2. **Add Item to Cart**
3. **View Cart**
4. **Checkout**
5. **Exit**

**Functions**

1. **View Available Items**
   * **Function Used:** **view\_items(items)**
   * **Description:** This function displays a list of all available grocery items. Each item is numbered for easy selection.
   * **Implementation Details:**
     + The function iterates through the **items** list.
     + Each item is printed with an index starting from 1 for user reference.
2. **Add Item to Cart**
   * **Function Used:** **add\_to\_cart(items, cart)**
   * **Description:** This function allows users to add items from the available list to their shopping cart.
   * **Implementation Details:**
     + The **view\_items(items)** function is called to display the items.
     + The user is prompted to enter the number corresponding to the item they wish to add to the cart.
     + The input is validated to ensure it is a digit and within the range of available items.
     + If valid, the selected item is added to the **cart** list, and a confirmation message is displayed.
     + If invalid, an error message is shown.
3. **View Cart**
   * **Function Used:** **view\_cart(cart)**
   * **Description:** This function displays the contents of the user's shopping cart.
   * **Implementation Details:**
     + If the cart is not empty, the items in the cart are listed.
     + If the cart is empty, a message indicating the cart is empty is displayed.
4. **Checkout**
   * **Function Used:** **checkout(cart)**
   * **Description:** This function simulates the checkout process for the items in the user's cart.
   * **Implementation Details:**
     + If the cart has items, a success message is displayed, and the cart is cleared.
     + If the cart is empty, a message indicating that checkout cannot proceed without items is shown.
5. **Exit**
   * **Description:** This option allows the user to exit the application.
   * **Implementation Details:**
     + A thank you message is displayed, and the main menu loop is terminated.

**Functions Used**

1. **display\_menu()**
   * **Purpose:** Displays the main menu with options for the user to choose from.
   * **Implementation:** Prints a list of menu options.
2. **view\_items(items)**
   * **Purpose:** Displays all available grocery items.
   * **Parameters:** **items** (list of strings representing grocery items)
   * **Implementation:** Iterates over the **items** list and prints each item with an index.
3. **add\_to\_cart(items, cart)**
   * **Purpose:** Adds a selected item to the user's shopping cart.
   * **Parameters:**
     + **items** (list of strings representing grocery items)
     + **cart** (list representing the user's cart)
   * **Implementation:**
     + Displays available items.
     + Prompts user for item number.
     + Validates input.
     + Adds item to **cart** if valid, otherwise shows an error.
4. **view\_cart(cart)**
   * **Purpose:** Displays the contents of the user's shopping cart.
   * **Parameters:** **cart** (list representing the user's cart)
   * **Implementation:**
     + Checks if **cart** is empty.
     + If not, prints each item in **cart**.
     + If empty, prints a message indicating the cart is empty.
5. **checkout(cart)**
   * **Purpose:** Simulates the checkout process.
   * **Parameters:** **cart** (list representing the user's cart)
   * **Implementation:**
     + Checks if **cart** is empty.
     + If not, displays a success message and clears **cart**.
     + If empty, prints a message indicating the cart is empty.
6. **gro()**
   * **Purpose:** Main function that runs the application.
   * **Implementation:**
     + Initializes the list of available items and an empty cart.
     + Continuously displays the menu and prompts the user for their choice.
     + Calls the appropriate function based on user input.
     + Exits the loop and application when the user selects the exit option.

**Medicine Delivery System**

The Medicine Delivery System is a command-line application designed to facilitate ordering and obtaining information about various medicines. It provides users with functionalities to order medicines, learn about them, and manage their shopping cart. The application is structured with an interactive menu, guiding users through the available options seamlessly.

**Functionalities**

1. **Order Medicine**
2. **Know About Medicine**
3. **Exit**

**Functions**

1. **Order Medicine**
   * **Description:** This function allows users to order medicines from a predefined list. Users can specify the quantity they need, and the system calculates the total cost, including a 10% additional charge.
   * **Implementation Details:**
     + Users are prompted to enter the name of the medicine they wish to order.
     + If the medicine is available, users are asked for the required quantity.
     + The selected medicine and quantity are added to the cart.
     + The total cost is calculated and displayed.
     + Users can continue adding more medicines or place the order.
2. **Know About Medicine**
   * **Description:** This function provides detailed information about the medicines available in the database. Users can inquire about specific medicines to learn about their uses and properties.
   * **Implementation Details:**
     + Users are prompted to enter the name of the medicine they want to know about.
     + If the medicine is in the database, detailed information is retrieved and displayed.
     + Users can continue querying for more medicines or return to the main menu.
3. **Exit**
   * **Description:** This option allows the user to exit the application.
   * **Implementation Details:**
     + When selected, the application exits and returns to the main selector function**.**

**Data Structures Used**

1. **mk (Medicine Keys):** A dictionary mapping medicine names to numerical keys representing different types of medicines.
2. **ml (Medicine List):** A dictionary mapping medicine names to their respective prices.
3. **mt (Medicine Types):** A dictionary mapping numerical keys to groups of medicines.
4. **mi (Medicine Information):** A dictionary mapping numerical keys to detailed descriptions of the medicines.

**Functions Used**

1. **med()**
   * **Purpose:** The main function that runs the Medicine Delivery System application.
   * **Implementation:**
     + Initializes the dictionaries containing medicine data.
     + Displays the main menu and prompts the user to choose an action.
     + Directs the user to the appropriate functionality based on their choice.
2. **Order Medicine Functionality (within med())**
   * **Purpose:** Allows users to order medicines and manage their cart.
   * **Implementation:**
     + Prompts the user for the name and quantity of the medicine.
     + Adds the selected medicine to the cart and calculates the total cost.
     + Allows users to continue ordering or place the order.
3. **Know About Medicine Functionality (within med())**
   * **Purpose:** Provides detailed information about the selected medicine.
   * **Implementation:**
     + Prompts the user for the name of the medicine.
     + Retrieves and displays the detailed information from the dictionary.
4. **Exit Functionality (within med())**
   * **Purpose:** Exits the application.
   * **Implementation:**
     + Calls the selector function to exit.

**Customer Support System**

The Customer Support System is a command-line application designed to facilitate interaction with customers needing assistance. It offers two main support options: chatting with an AI bot (CHERA) or chatting with a human assistant. The application also includes various sub-functionalities for different types of queries, ensuring comprehensive support for users.

**Functionalities**

1. **Chat with CHERA (AI Bot)**
2. **Chat with Human Assistant**
3. **Go Back to Main Menu**

**Explanation of Functionalities**

1. **Chat with CHERA (AI Bot)**
   * **Description:** This function allows users to interact with an AI bot for common queries related to general information, medicine availability, orders, and payments/refunds.
   * **Implementation Details:**
     + Users can choose from several categories of queries.
     + Based on the user's choice, the AI bot provides predefined responses or requests further information.
2. **Chat with Human Assistant**
   * **Description:** This function connects users to a human representative for more personalized support.
   * **Implementation Details:**
     + Users are prompted to enter their message.
     + The system acknowledges the message and informs the user that they are being connected to a human assistant.
3. **Go Back to Main Menu**
   * **Description:** This option allows the user to exit the support system and return to the main menu of the application.
   * **Implementation Details:**
     + When selected, the application calls the main selector function to navigate back.

**Functions Used**

1. **supp()**
   * **Purpose:** The main function that runs the Customer Support System application.
   * **Implementation:**
     + Displays the support menu and prompts the user to choose an action.
     + Calls the **con()** function with the user's choice.
2. **con(opt)**
   * **Purpose:** Routes the user's choice to the appropriate support function.
   * **Implementation:**
     + Based on the input **opt**, it calls either the **ai()**, **hum()**, or **selector()** function.
3. **hum()**
   * **Purpose:** Simulates a chat with a human assistant.
   * **Implementation:**
     + Prompts the user to enter their message.
     + Provides a standard response indicating that a human representative will be in touch.
4. **ai()**
   * **Purpose:** Provides interaction with the AI bot for various queries.
   * **Implementation:**
     + Displays a menu of query categories.
     + Routes the user's choice to the appropriate function: **gq()**, **am()**, **orde()**, or **pr()**.
5. **gq()**
   * **Purpose:** Handles general queries.
   * **Implementation:**
     + Displays a menu of general questions.
     + Provides predefined responses based on the user's choice.
6. **am()**
   * **Purpose:** Handles queries related to the availability of medicines.
   * **Implementation:**
     + Displays a menu of availability-related questions.
     + Checks the medicine database and provides responses based on availability and alternatives.
7. **orde()**
   * **Purpose:** Handles queries related to orders.
   * **Implementation:**
     + Displays a menu of order-related questions.
     + Provides predefined responses regarding order status, delivery, payment confirmation, and order modifications.
8. **pr()**
   * **Purpose:** Handles queries related to payment or refund issues.
   * **Implementation:**
     + Displays a menu of payment/refund-related questions.
     + Provides predefined responses for double charges, failed payments, accidental orders, product returns, and overcharges.

**Data Structures Used**

1. **mk (Medicine Keys):** A dictionary mapping medicine names to numerical keys representing different types of medicines.
2. **ml (Medicine List):** A dictionary mapping medicine names to their respective prices.
3. **mt (Medicine Types):** A dictionary mapping numerical keys to groups of medicines.
4. **mi (Medicine Information):** A dictionary mapping numerical keys to detailed descriptions of the medicines.

**Potential improvements:**

* **Error handling:** Consider adding error handling for invalid user inputs and potential exceptions during calculations.
* **Security:** For a real-world application, you'd want to implement more secure password storage using hashing algorithms.
* **Database:** Instead of using dictionaries, a database can be used to store user information, orders, and product details for better scalability and data persistence.
* **Payment Integration:** Integrate a payment gateway for secure online transactions.
* **User Interface:** This code is designed for a text-based interface. You can extend it to create a graphical user interface (GUI) for a more user-friendly experience.