****

**Projects – Week 5**

**Project 1**

Simple Inventory Management System

**Objective**

Create a basic text-based inventory management system that allows users to add, update, delete, and view products. Utilize Python basics such as dictionaries, loops, and conditional statements to create the application.

**Requirements**

1. Use dictionaries to store product information, such as product IDs, names, quantities, and prices.
2. Use loops to create a menu-driven interface for users to interact with the system.
3. Use conditional statements to perform various operations, such as adding, updating, deleting, and viewing products.

**Instructions**

1. Create a dictionary to store product information, including product IDs, names, quantities, and prices.
2. Design a menu-driven interface that allows users to choose between the following operations:
   1. Add a product
   2. Update a product
   3. Delete a product
   4. View all products
   5. Exit
3. Use a loop to iterate through the menu, allowing users to perform multiple operations without having to restart the application.
4. Based on the user's selection, use conditional statements to perform the chosen operation:
   1. Add a product: Request the necessary product information from the user and add it to the dictionary.
   2. Update a product: Ask the user for the product ID, find the product in the dictionary, and update the relevant information.
   3. Delete a product: Ask the user for the product ID and remove the product from the dictionary.
   4. View all products: Display the product information stored in the dictionary.
5. Continue looping through the menu until the user chooses to exit the application.

**Example**

Menu:

1. Add a product
2. Update a product
3. Delete a product
4. View all products
5. Exit

Please choose an option (1-5): 1

Enter product ID: 101

Enter product name: Laptop

Enter quantity: 5

Enter price: 1200

**Project 2**

Simple Task Manager

**Objective**

Develop a simple task manager where users can create, update, delete, and view tasks. Utilize Python basics such as dictionaries, loops, and conditional statements to create the application.

**Requirements**

1. Use dictionaries to store task information, such as task IDs, titles, descriptions, and deadlines.
2. Use loops to create a menu-driven interface for users to interact with the system.
3. Use conditional statements to perform various operations, such as creating, updating, deleting, and viewing tasks.

**Instructions**

1. Create a dictionary to store task information, including task IDs, titles, descriptions, and deadlines.
2. Design a menu-driven interface that allows users to choose between the following operations:
   1. Create a task
   2. Update a task
   3. Delete a task
   4. View all tasks
   5. Exit
3. Use a loop to iterate through the menu, allowing users to perform multiple operations without having to restart the application.
4. Based on the user's selection, use conditional statements to perform the chosen operation:
   1. Create a task: Request the necessary task information from the user and add it to the dictionary.
   2. Update a task: Ask the user for the task ID, find the task in the dictionary, and update the relevant information.
   3. Delete a task: Ask the user for the task ID and remove the task from the dictionary.
   4. View all tasks: Display the task information stored in the dictionary.
5. Continue looping through the menu until the user chooses to exit the application.

**Example**

Menu:

1. Create a task
2. Update a task
3. Delete a task
4. View all tasks
5. Exit

Please choose an option (1-5): 1

Enter task ID: 1001

Enter task title: Complete Python project

Enter task description: Finish the simple task manager project for class

Enter task deadline: 2023-04-10

**Project 3**

Text-Based Voting System

**Objective**

Create a text-based voting system that allows users to vote for their favorite candidates in a mock election. Utilize Python basics such as dictionaries, loops, and conditional statements to create the application.

**Requirements**

1. Use dictionaries to store candidate names and vote counts.
2. Use loops to iterate through the list of candidates and update vote counts.
3. Use conditional statements to manage voting and display the results.
4. Display the results, including the total votes for each candidate and the winner.

**Instructions**

1. Create a dictionary to store candidate names and their corresponding vote counts.
2. Display the list of candidates to the user.
3. Ask the user to vote for their favorite candidate by entering the candidate's name.
4. Use a loop to iterate through the list of candidates, updating the vote count for the chosen candidate using conditional statements.
5. Allow users to vote multiple times by using a loop that continues until the user decides to stop voting.
6. After the voting is complete, display the results by iterating through the dictionary and showing the total votes for each candidate.
7. Determine the winner by comparing the vote counts of each candidate using loops and conditional statements.
8. Display the winner of the mock election.

**Example**

Candidates:

1. Alice
2. Bob
3. Carol

Enter the name of the candidate you want to vote for (or type 'end' to finish voting): Alice

Keep voting until the user types **'end**'.

Results:

Alice: 10 votes

Bob: 7 votes

Carol: 8 votes

Winner: Alice

**Project 4**

Simple Movie Database

**Objective**

Create a simple movie database that allows users to add, update, delete, and view movie information. Utilize Python basics such as dictionaries, loops, and conditional statements to create the application.

**Requirements**

1. Use dictionaries to store movie information, such as titles, directors, release years, and genres.
2. Use loops to create a menu-driven interface for users to interact with the database.
3. Use conditional statements to perform various operations, such as adding, updating, deleting, and viewing movies.

**Instructions**

1. Create a dictionary to store movie information, including titles, directors, release years, and genres.
2. Design a menu-driven interface that allows users to choose between the following operations:
   1. Add a movie
   2. Update a movie
   3. Delete a movie
   4. View all movies
   5. Exit
3. Use a loop to iterate through the menu, allowing users to perform multiple operations without having to restart the application.
4. Based on the user's selection, use conditional statements to perform the chosen operation:
   1. Add a movie: Request the necessary movie information from the user and add it to the dictionary.
   2. Update a movie: Ask the user for the movie title, find the movie in the dictionary, and update the relevant information.
   3. Delete a movie: Ask the user for the movie title and remove the movie from the dictionary.
   4. View all movies: Display the movie information stored in the dictionary.
5. Continue looping through the menu until the user chooses to exit the application.

**Example**

Menu:

1. Add a movie
2. Update a movie
3. Delete a movie
4. View all movies
5. Exit

Please choose an option (1-5): 1

Enter movie title: The Shawshank Redemption

Enter director: Frank Darabont

Enter release year: 1994

Enter genre: Drama

**Project 5**

Simple Text-Based Restaurant Menu and Order System

**Objective**

Develop a simple text-based restaurant menu and order system that allows users to view the menu, add items to their order, and calculate the total cost. Utilize Python basics such as dictionaries, loops, and conditional statements to create the application.

**Requirements**

1. Use dictionaries to store meunu items and their prices.
2. Use loops to create a menu-driven interface for users to interact with the system.
3. Use conditional statements to perform various operations, such as viewing the menu, adding items to the order, and calculating the total cost.

**Instructions**

1. Create a dictionary to store menu items and their prices.
2. Design a menu-driven interface that allows users to choose between the following operations:
   1. View the menu
   2. Add an item to the order
   3. Calculate the total cost
   4. Exit
3. Use a loop to iterate through the menu, allowing users to perform multiple operations without having to restart the application.
4. Based on the user's selection, use conditional statements to perform the chosen operation:
   1. View the menu: Display the menu items and their prices stored in the dictionary.
   2. Add an item to the order: Ask the user for the menu item they want to add to their order and store it in a list.
   3. Calculate the total cost: Iterate through the list of ordered items and calculate the total cost using the prices stored in the dictionary.
5. Continue looping through the menu until the user chooses to exit the application.

Menu:

1. View the menu
2. Add an item to the order
3. Calculate the total cost
4. Exit

Please choose an option (1-4): 1

Menu Items:

Burger: 5.00

Pizza: 10.00

Salad: 7.00

**Project 6**

Food Delivery Tracking System

**Description**

The Food Delivery Tracking System is a text-based application that allows users to track the status of their food delivery in real-time. It utilizes dictionaries, loops, and conditional statements to manage and display delivery order information.

**Features**

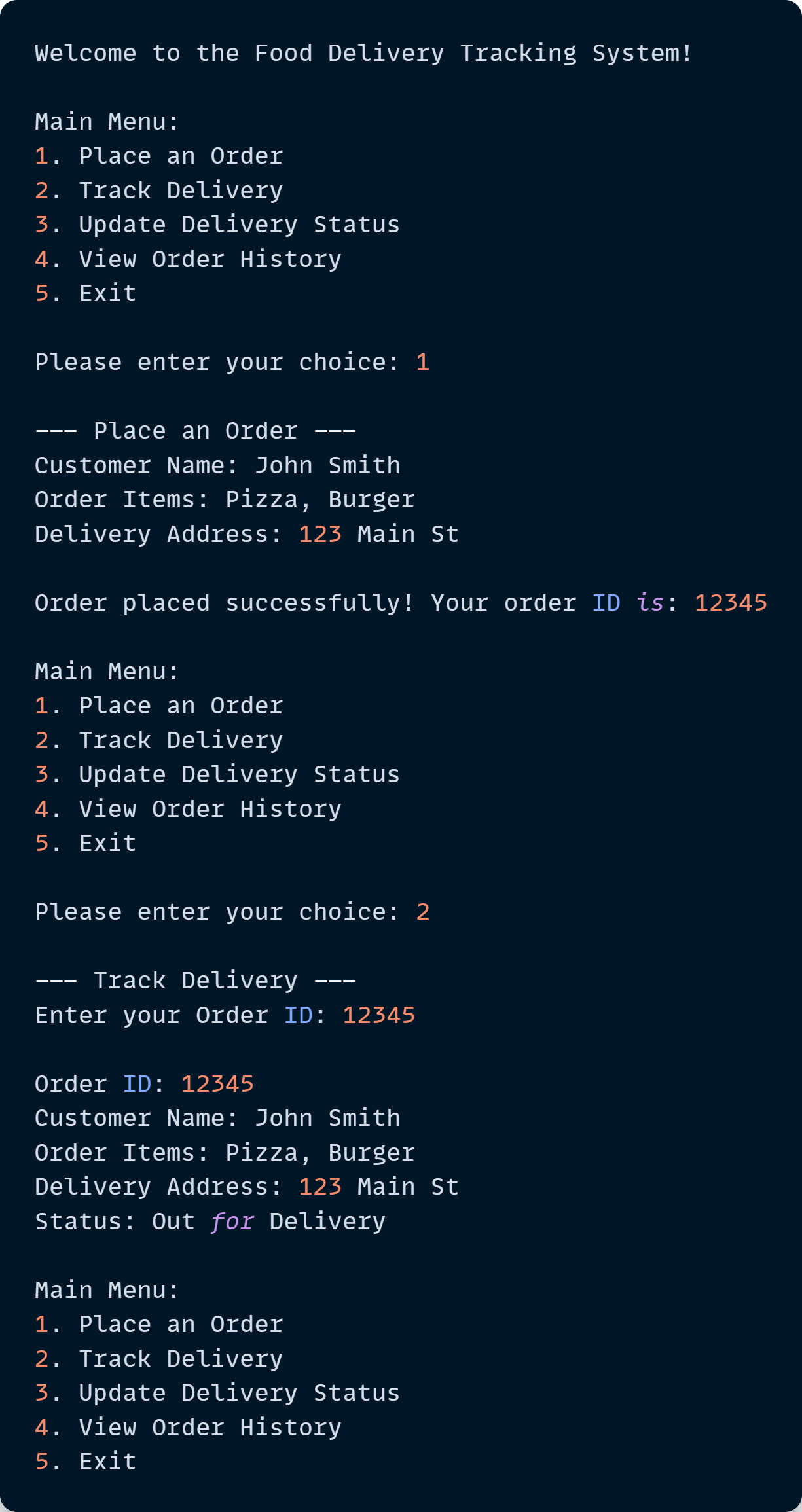
1. Place an Order: Users can input their order details, including the customer name, order items, and delivery address.
2. Track Delivery: Users can enter their order ID to track the status of their delivery.
3. Update Delivery Status: Delivery personnel can update the status of each order, such as "Preparing", "Out for Delivery", and "Delivered".
4. View Order History: Users can view the order history and delivery details of their previous orders.

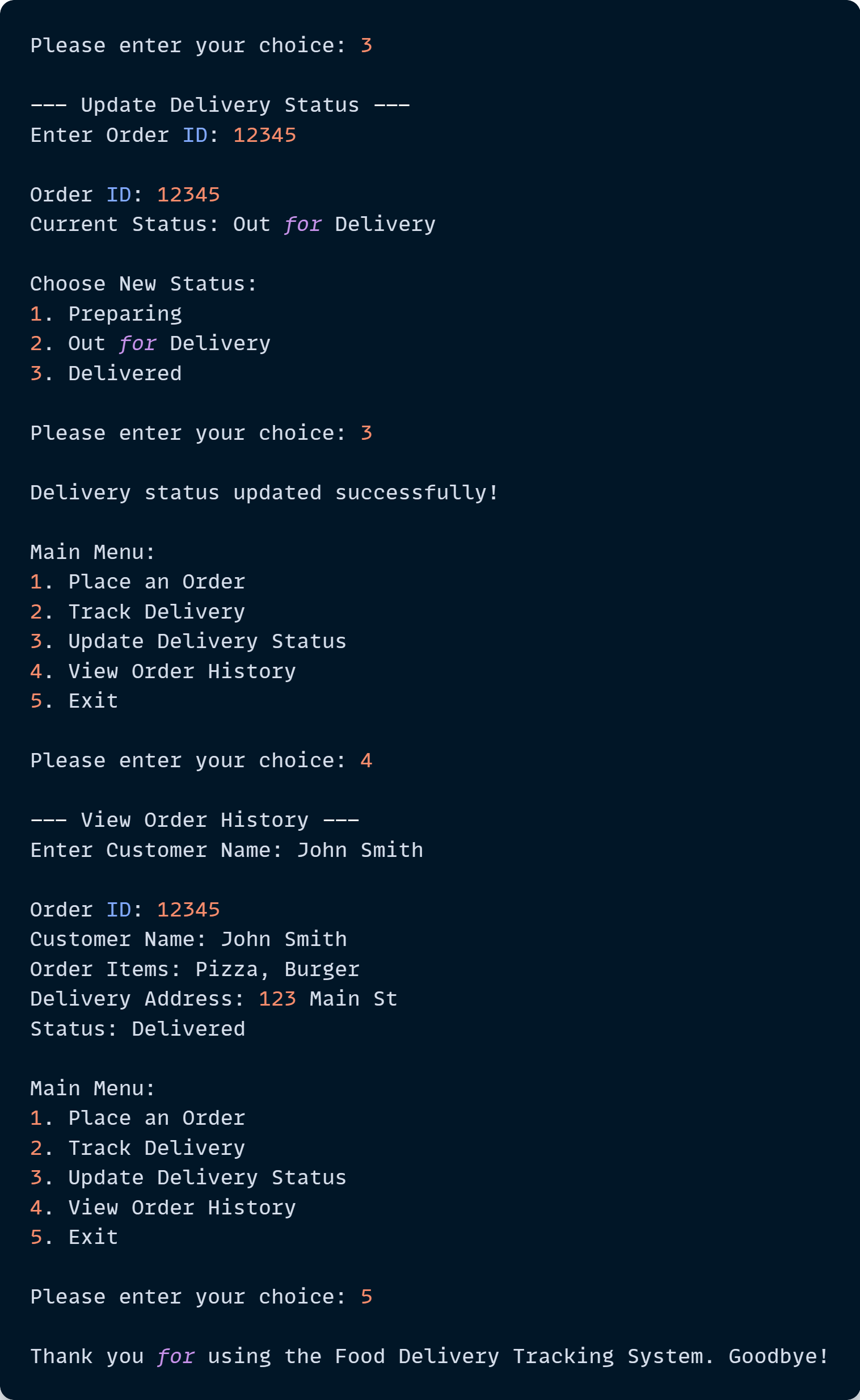
**Implementation Details:**

1. Data Structure: The system utilizes dictionaries to store delivery order information. Each order is represented by a unique order ID, and its details include the customer name, order items, delivery address, and status.

Example: order\_id -> {'customer\_name': 'John Smith', 'order\_items': ['Pizza', 'Burger'], 'delivery\_address': '123 Main St', 'status': 'Out for Delivery'}

1. Main Menu: The system presents a menu-driven interface to users with the following options:
   1. Place an Order
   2. Track Delivery
   3. Update Delivery Status
   4. View Order History
   5. Exit
2. Place an order:
   * The user enters their name, order items (separated by commas), and delivery address.
   * A new entry is added to the dictionary, using a unique order ID as the key and the order details as the value.
   * The order status is initially set to "Preparing".
3. Track Delivery:
   * The user enters their order ID.
   * The system retrieves the corresponding order details from the dictionary.
   * The current status of the order is displayed.
4. Update delivery status:
   * The delivery personnel enters the order ID.
   * The system retrieves the order details and displays the current status.
   * The personnel can update the status by selecting from predefined options (e.g., "Preparing", "Out for Delivery", "Delivered").
   * The updated status is stored back in the dictionary.
5. View order history:
   * The user enters their name or customer ID.
   * The system retrieves all orders associated with the user from the dictionary and displays their details.
6. Exit:
   * The user chooses to exit the application, ending the program execution.

****

****

**Sample Project**

Simple Expense Tracker

**Objective**

Develop a simple expense tracker that allows users to add, update, delete, and view expenses. Utilize Python basics such as dictionaries, loops, and conditional statements to create the application.

**Requirements**

1. Use dictionaries to store expense information, such as expense IDs, descriptions, categories, and amounts.
2. Use loops to create a menu-driven interface for users to interact with the tracker.
3. Use conditional statements to perform various operations, such as adding, updating, deleting, and viewing expenses.

**Instructions**

1. Create a dictionary to store expense information, including expense IDs, descriptions, categories, and amounts.
2. Design a menu-driven interface that allows users to choose between the following operations:
   1. Add an expense
   2. Update an expense
   3. Delete an expense
   4. View all expenses
   5. Exit
3. Use a loop to iterate through the menu, allowing users to perform multiple operations without having to restart the application.
4. Based on the user's selection, use conditional statements to perform the chosen operation:
   1. Add an expense: Request the necessary expense information from the user and add it to the dictionary.
   2. Update an expense: Ask the user for the expense ID, find the expense in the dictionary, and update the relevant information.
   3. Delete an expense: Ask the user for the expense ID and remove the expense from the dictionary.
   4. View all expenses: Display the expense information stored in the dictionary.
5. Continue looping through the menu until the user chooses to exit the application.



