# MICRO PROJECT REPORT

# (Data Structure with Python - 4331601)

# Submitted By

Mahil Parmar (23640031106)

Devraj Patel (236400316116)

Jenish Patel (236400316131)

Maharshi Patel (236400316135)

In partial fulfilment for the curriculum of the 3rd Semester in

DIPLOMA ENGINEERING in

# Information Technology Department



R. C. Technical Institute, Ahmedabad

Gujarat Technological University, Ahmedabad

# July, 2024

**Index**

|  |  |
| --- | --- |
| **Content** | **Page no.** |
| Introduction | 1 |
| Features and Functionality | 2-3 |
| Code | 4 |
| Code explanation | 5-6 |
| Output | 7-8 |
| Conclusion | 9 |

# **INTRODUCTION**

“Ticket Management System” is a very widely used system nowadays. Implementation of this system using python on very small scale has been tried to carry out. This system works to distribute multiple types of tickets to users with user efficiency. This project is firmly made without using any AI tool. However an overview of the title and explanation was taken for reference.

The “Ticket Management System” is designed to facilitate the process of ticket booking, allocation, and management for events or transportation services. With various ticket categories such as VIP, General, and Student, each having its own price and limited availability, the system aims to streamline the user experience while managing ticket inventory efficiently.

This system allows users to request tickets by specifying the type and quantity. It manages these requests through a queue, ensuring that each user’s request is processed in the order received. Users can also view the available tickets, which provides real-time information on the remaining quantities for each type of ticket.

The primary objectives of this project include:

* **Efficient Request Handling**: To allow users to make multiple ticket requests and handle them in a queue.
* **Inventory Management**: To track the number of tickets available for each category and prevent overselling by validating user input against availability.
* **Ticket Processing**: To process user requests and allocate tickets accordingly, ensuring that requests are properly managed before final allocation.
* **User-Friendly Interaction**: To provide an easy-to-use command-line interface that prompts users to make requests, finalize tickets, and view ticket availability.

By automating the process of ticket management, the system reduces the manual effort required to handle bookings and ensures accurate real-time tracking of ticket inventory. It also addresses common issues like overbooking by automatically checking ticket availability before confirming any request.

This project is written in Python, utilizing fundamental programming concepts like classes, lists, dictionaries, and user input handling. It emphasizes simplicity while providing core functionalities that could be expanded in the future with additional features such as online payments or graphical user interfaces (GUIs).

1.

**Features and Functionalities**

#### **1.** **Ticket Request Handling**

* Users can request tickets for three different types: **VIP**, **General**, and **Student**, each with distinct prices and availability.
* **Validation of Input**: The system ensures that:
  + The ticket type entered by the user is valid (must be one of the available types: VIP, General, or Student).
  + The quantity requested does not exceed the number of available tickets for that type.
  + If the requested quantity is greater than available tickets, the system displays an error message: “Insufficient tickets requested!!”
* Upon successful validation, the system adds the ticket request to a **queue** (ticket\_queue). Each request in the queue contains:
  + The ticket type
  + The price of the ticket
  + The quantity requested
* **Pending Requests Limitation**: If there are already five pending requests in the queue, the system asks the user to finalize the existing requests before adding new ones. This prevents the user from overwhelming the system with too many unprocessed requests.

#### **2.** **Ticket Processing and Allocation**

* After requesting tickets, users can choose to **process** the requests.
* The system processes the queue in a **first-come, first-served** manner.
* The user is prompted to enter their **name**, and the system allocates the requested tickets to the user. For each request in the queue:
  + The system outputs a message showing the number of tickets and their type allocated to the user.
* Once the tickets are allocated, the system removes the processed request from the queue, clearing space for new requests.
* If the user attempts to process tickets when there are no requests in the queue, the system displays the message: “Please request a ticket first”.

#### **3.** **Displaying Available Tickets**

* Users can check the availability of tickets at any time by selecting the display option.
* The system displays the **current availability** of each ticket type, including:
  + **Ticket Type**: VIP, General, or Student
  + **Price**: The price per ticket for each category
  + **Quantity Available**: The number of tickets still available for each category

2.

#### **4.** **User-Friendly Command-Line Interface**

* The system provides an easy-to-use **command-line interface** (CLI) that allows users to interact with the system through a set of simple prompts and commands.
* The main menu presents users with the following options:
  + Press **1** to request tickets.
  + Press **2** to finalize and process ticket allocations.
  + Press **3** to display the currently available tickets.
  + Press **0** to exit the system.
* Invalid inputs (such as entering non-existent ticket types or incorrect menu options) are handled gracefully, with the system displaying error messages to guide users back to valid choices.

#### **5.** **Real-Time Inventory Management**

* The system tracks the available ticket inventory in real-time. Every time a user requests tickets, the system automatically updates the ticket availability.
* The system prevents **overselling** by ensuring that users cannot request more tickets than are currently available. It also dynamically reduces the available ticket count when requests are confirmed, providing an up-to-date view of ticket availability at all times.

#### **6.** **Error Handling**

* The system implements robust error handling to manage various input-related issues:
  + If the user enters an invalid ticket type, the system prompts them to re-enter the correct type.
  + If the user requests an invalid quantity (e.g., a non-integer or a quantity larger than available), the system returns an appropriate error message.
  + The system ensures that invalid or incomplete inputs (such as leaving required fields blank) are handled gracefully without crashing or causing logical errors.

#### **7.** **Queue System for Request Management**

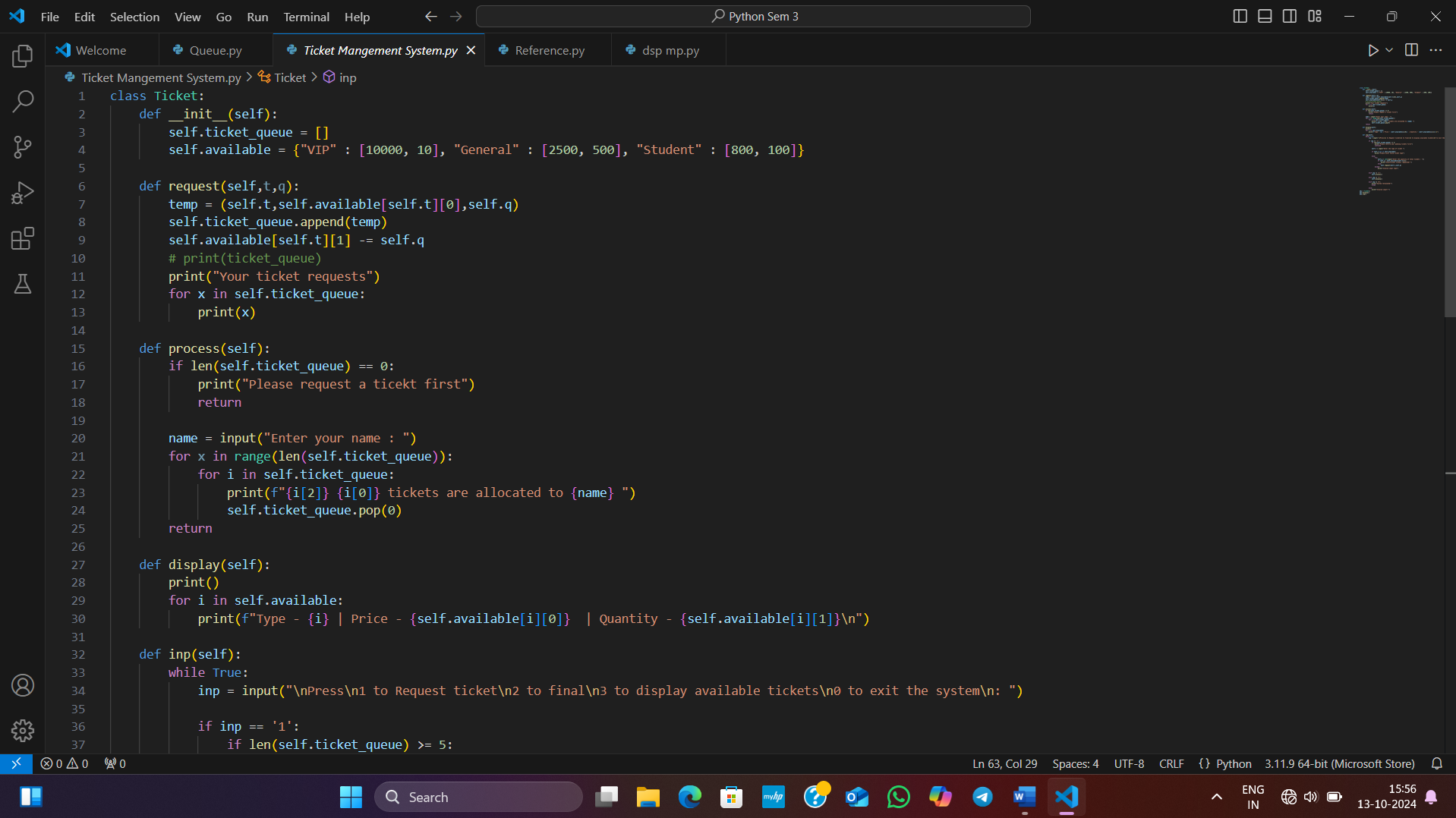
* The system employs a **queue** to manage multiple ticket requests, ensuring that tickets are processed in the order in which they are requested.
* The queue can hold up to five pending requests at a time, requiring users to finalize those requests before adding new ones.

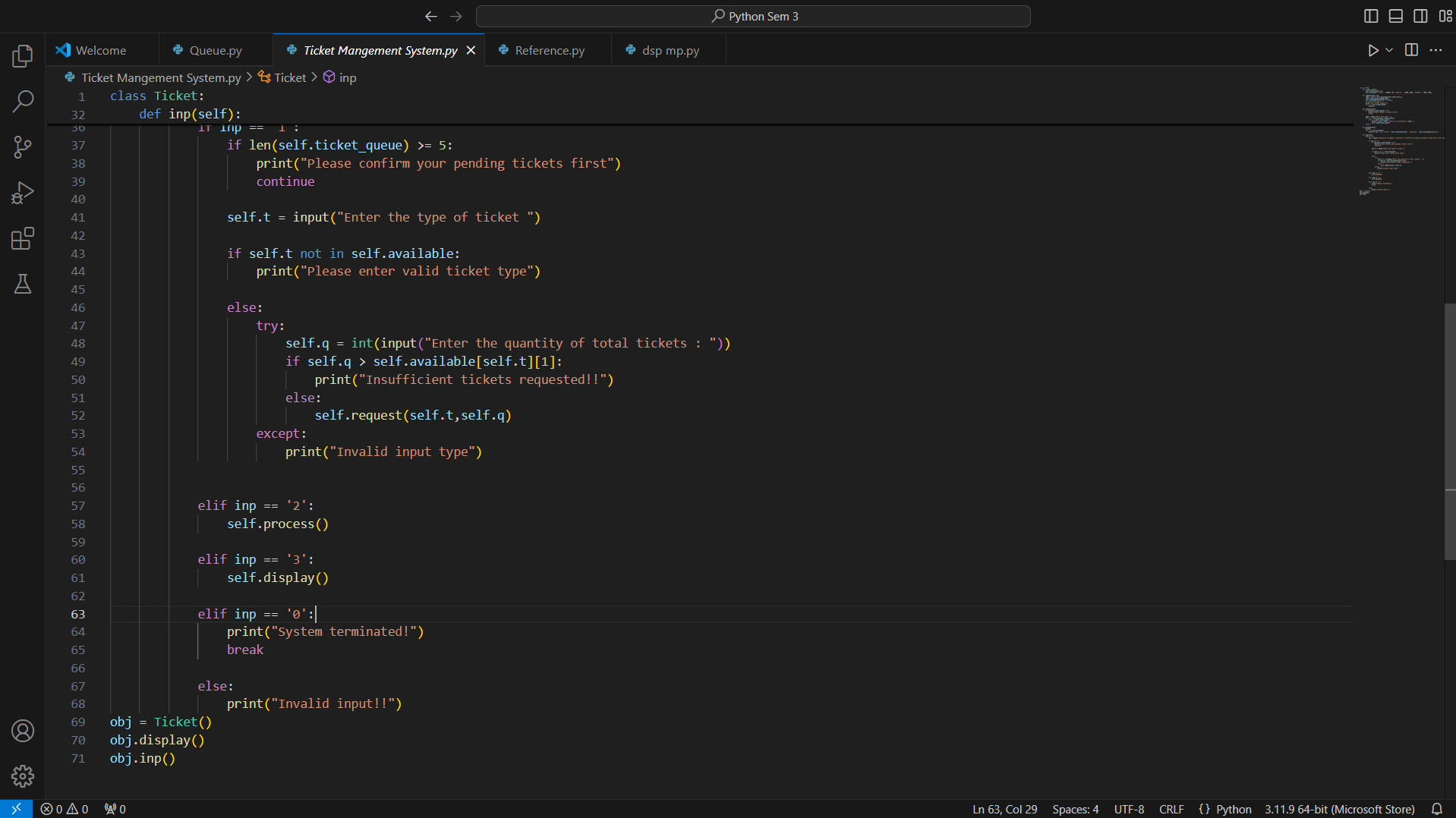
#### **8.** **System Termination**

* The system allows users to exit at any time by selecting the **exit option (0)** from the main menu. Upon exiting, the system displays a message confirming that it has been terminated.

3.

**CODE**





4.

**CODE EXPLANATION**

#### 1. **Class: Ticket**

This class handles all the ticket management functionalities:

* **Attributes**:
  + ticket\_queue: A list that stores all pending ticket requests made by users.
  + available: A dictionary that keeps track of the types of tickets (VIP, General, Student), their prices, and the number of tickets available for each type.

#### **2. Method: \_\_init\_\_()**

The constructor initializes the key attributes:

**def \_\_init\_\_(self):**

**self.ticket\_queue = []**

**self.available = {"VIP" : [10000, 10], "General" : [2500, 500], "Student" : [800, 100]}**

* ticket\_queue starts as an empty list to hold ticket requests.
* available is a dictionary with ticket types as keys. Each ticket type has an associated list where:
  + The first element is the ticket price.
  + The second element is the quantity of tickets available.

#### **3. Method: request(t, q)**

This method processes a user’s ticket request:

**def request(self, t, q):**

**temp = (self.t, self.available[self.t][0], self.q)**

**self.ticket\_queue.append(temp)**

**self.available[self.t][1] -= self.q**

**print("Your ticket requests")**

**for x in self.ticket\_queue:**

**print(x)**

* **Parameters**:
  + t: Type of ticket (VIP, General, Student).
  + q: Quantity of tickets requested.
* **Process**:
  + A tuple (temp) is created that holds the ticket type, price, and quantity requested.
  + The request is added to ticket\_queue.
  + The available ticket quantity is reduced by the requested amount.
  + Finally, the updated ticket queue is displayed, showing all pending requests.

5.

#### **4. Method: process()**

This method allocates tickets to the user based on their requests:

**def process(self):**

**if len(self.ticket\_queue) == 0:**

**print("Please request a ticket first")**

**return**

**name = input("Enter your name: ")**

**for x in range(len(self.ticket\_queue)):**

**for i in self.ticket\_queue:**

**print(f"{i[2]} {i[0]} tickets are allocated to {name}")**

**self.ticket\_queue.pop(0)**

**return**

* If the queue is empty, the system prompts the user to request tickets first.
* The system processes each ticket request in the queue, allocating the tickets to the user, and removes the processed requests from ticket\_queue using pop().

#### **5. Method: display()**

Displays the current availability of all ticket types:

**def display(self):**

**print()**

**for i in self.available:**

**print(f"Type - {i} | Price - {self.available[i][0]} | Quantity - {self.available[i][1]}\n")**

* For each ticket type in the available dictionary, the ticket’s type, price, and remaining quantity are displayed.

#### **6. Method: inp()**

This is the main interactive loop that provides the user with options:

**def inp(self):**

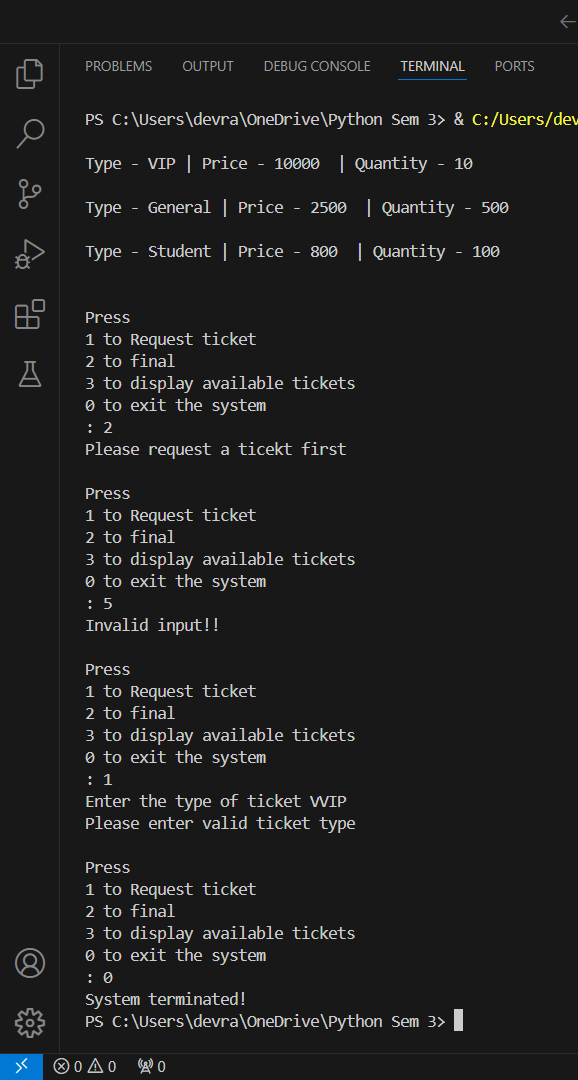
**while True:**

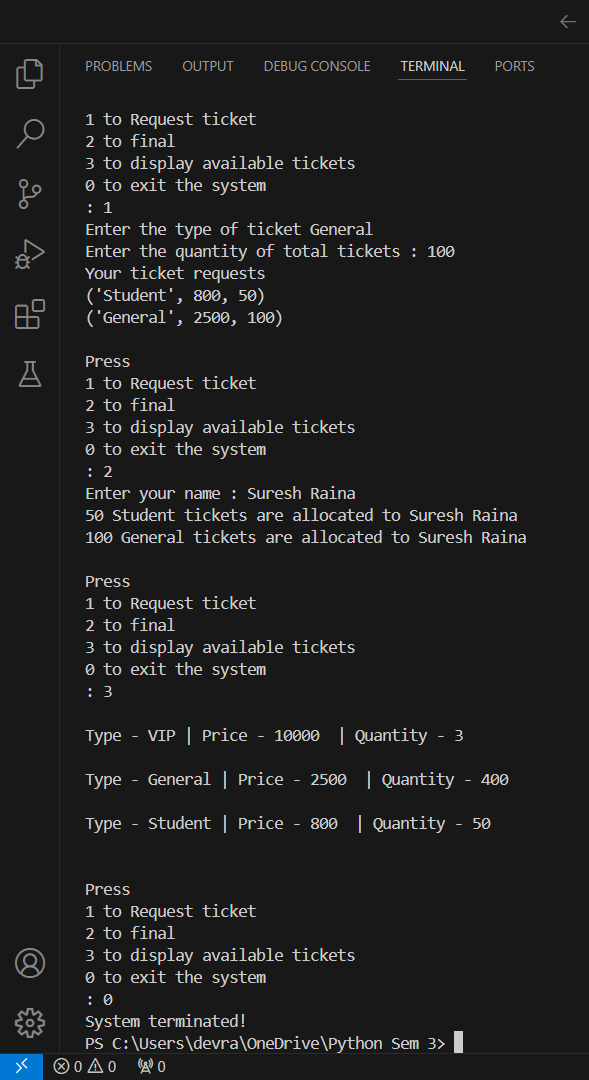
**inp = input("\nPress\n1 to Request ticket\n2 to final\n3 to display available tickets\n0 to exit the system\n: ")**

* **Handling User Inputs**:
  + If the user chooses to request a ticket (inp == '1'):
    - The system validates the ticket type and quantity.
    - If valid, the request is processed using the request() method.
    - If invalid, an error message is shown.
  + If the user selects **2**, the process() method is called to allocate the requested tickets.
  + If the user selects **3**, the display() method shows the available tickets.
  + The system will exit if **0** is pressed

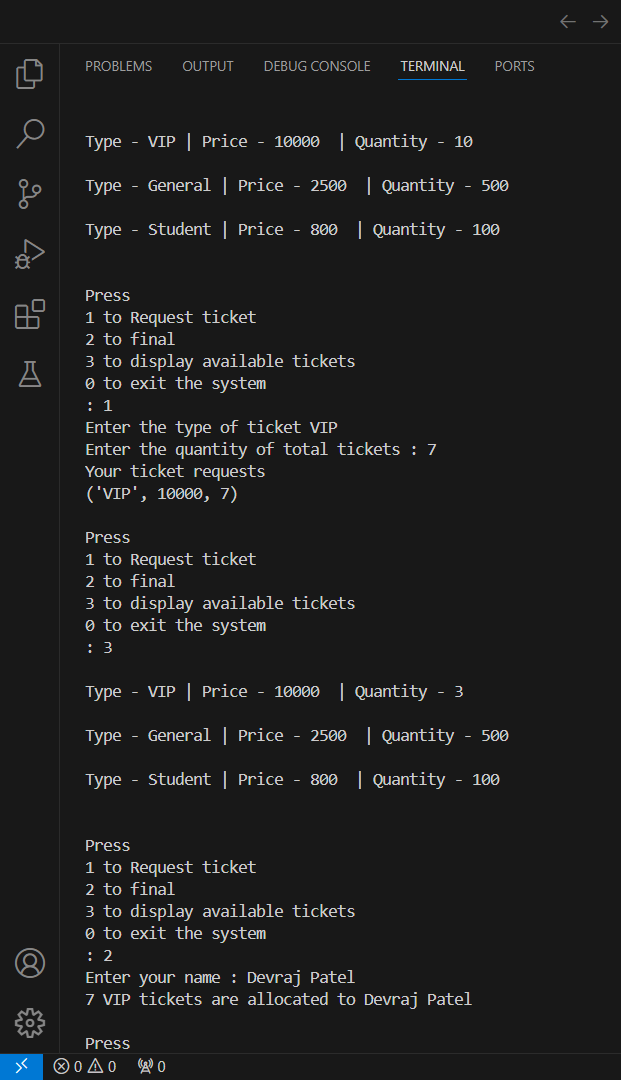
6.

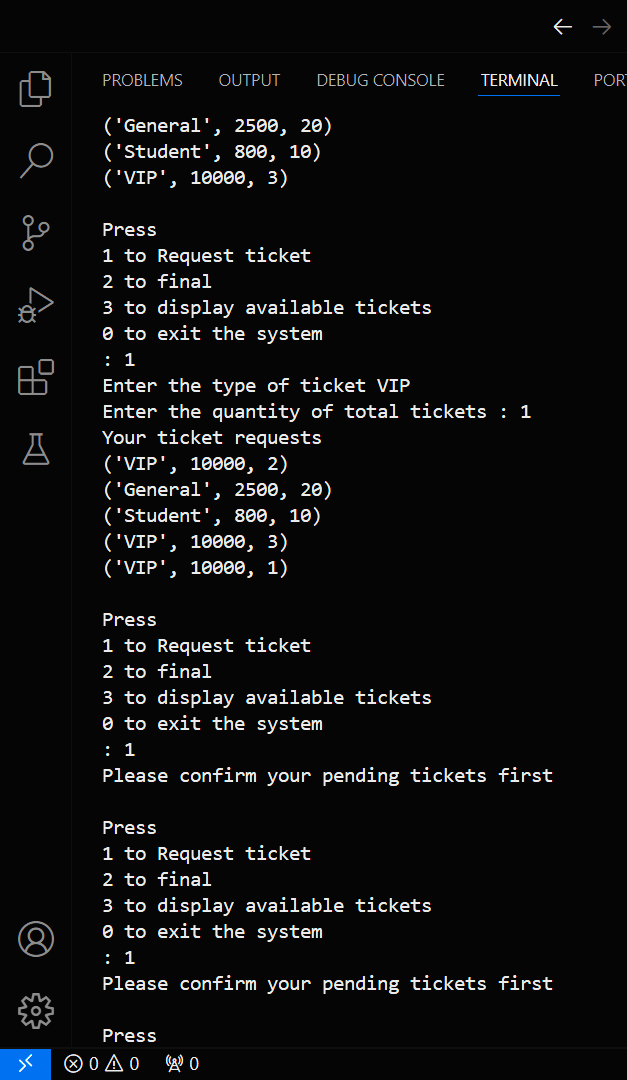
**Console Output**





7.





8.

**CONCLUSION**

This project demonstrates a practical application of Python concepts like class design, dictionary manipulation, and error handling, offering a user-friendly experience for managing event tickets. The system is robust in handling multiple user inputs and scenarios, such as invalid ticket requests or excessive ticket demands, while ensuring that tickets are allocated efficiently.

In conclusion, the Ticket Management System serves as a foundation for learning Python programming in a real-world scenario, with scope for further enhancements, such as adding user authentication, integrating payments, or expanding the range of ticket types. This project offers a great example of building scalable, interactive command-line applications.

9.