ASSIGNMENT- 11.3

Name:E.POOJITHA

HT.No: 2303A51356

Batch: 20

Task 1: Smart Contact Manager (Arrays & Linked Lists)

Sample input

1. Implement the contact manager using arrays (lists).

2. Implement the same functionality using a linked list for dynamic memory allocation.

3. Implement the following operations in both approaches:

o Add a contact

o Search for a contact

o Delete a contact

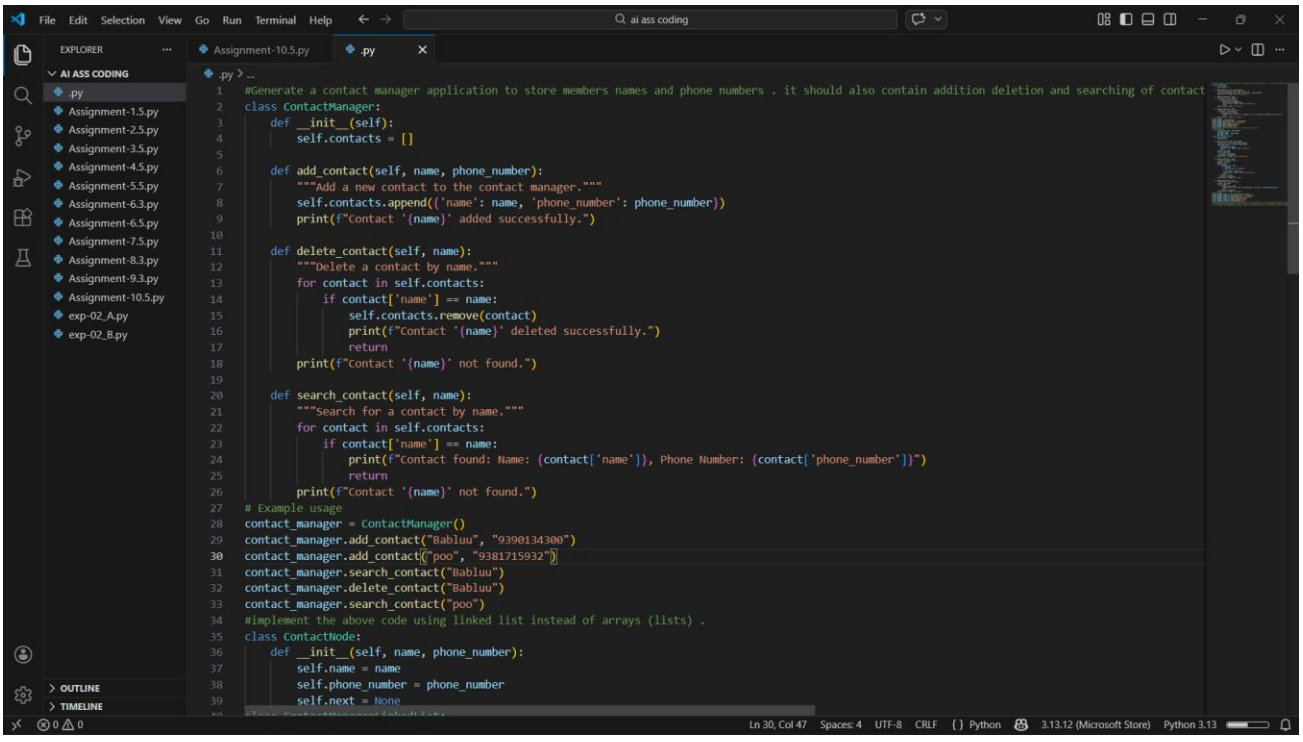
4. Use GitHub Copilot to assist in generating search and delete methods.

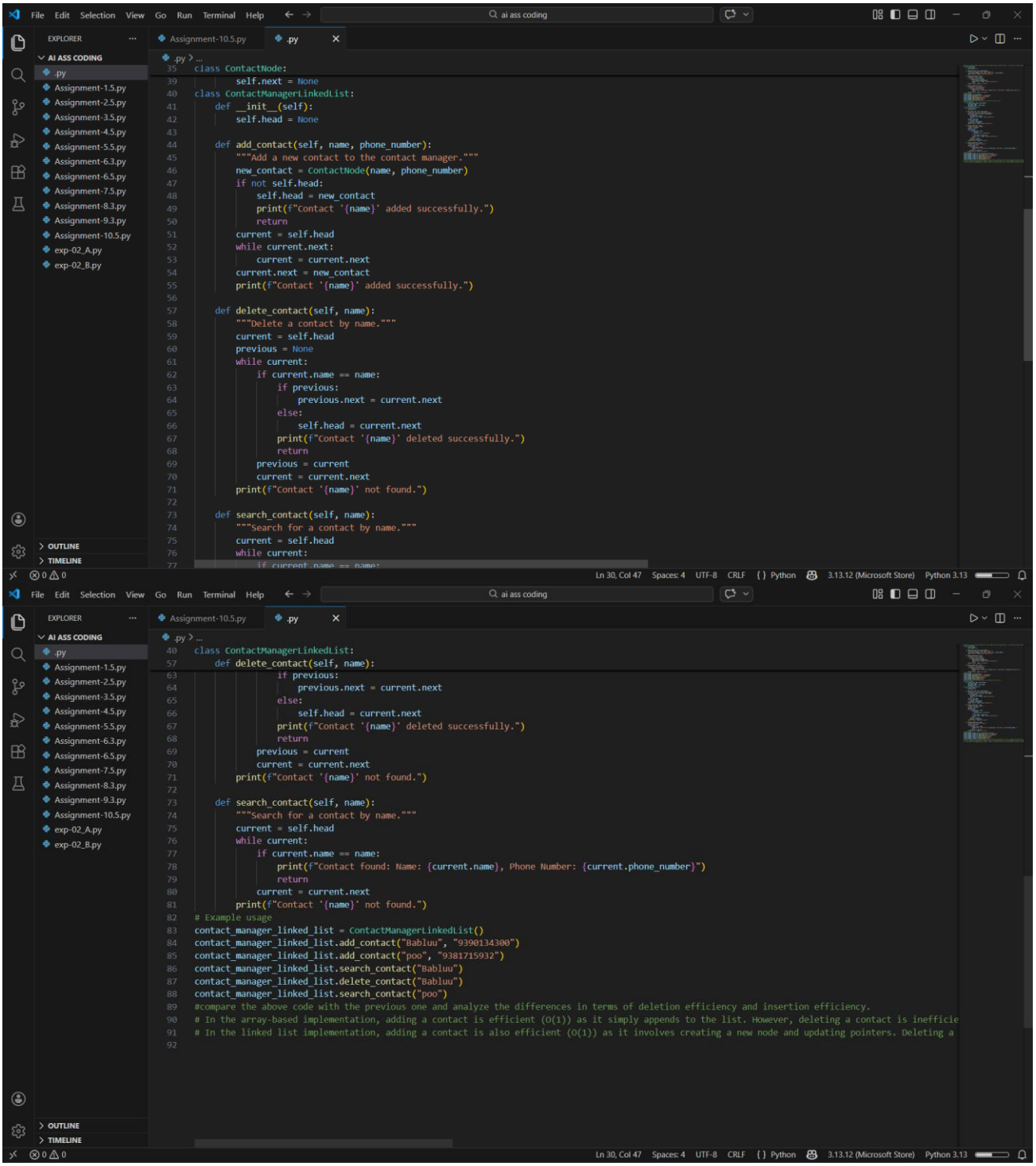
5. Compare array vs. linked list approaches with respect to:

o Insertion efficiency

o Deletion efficiency

Code & probmt

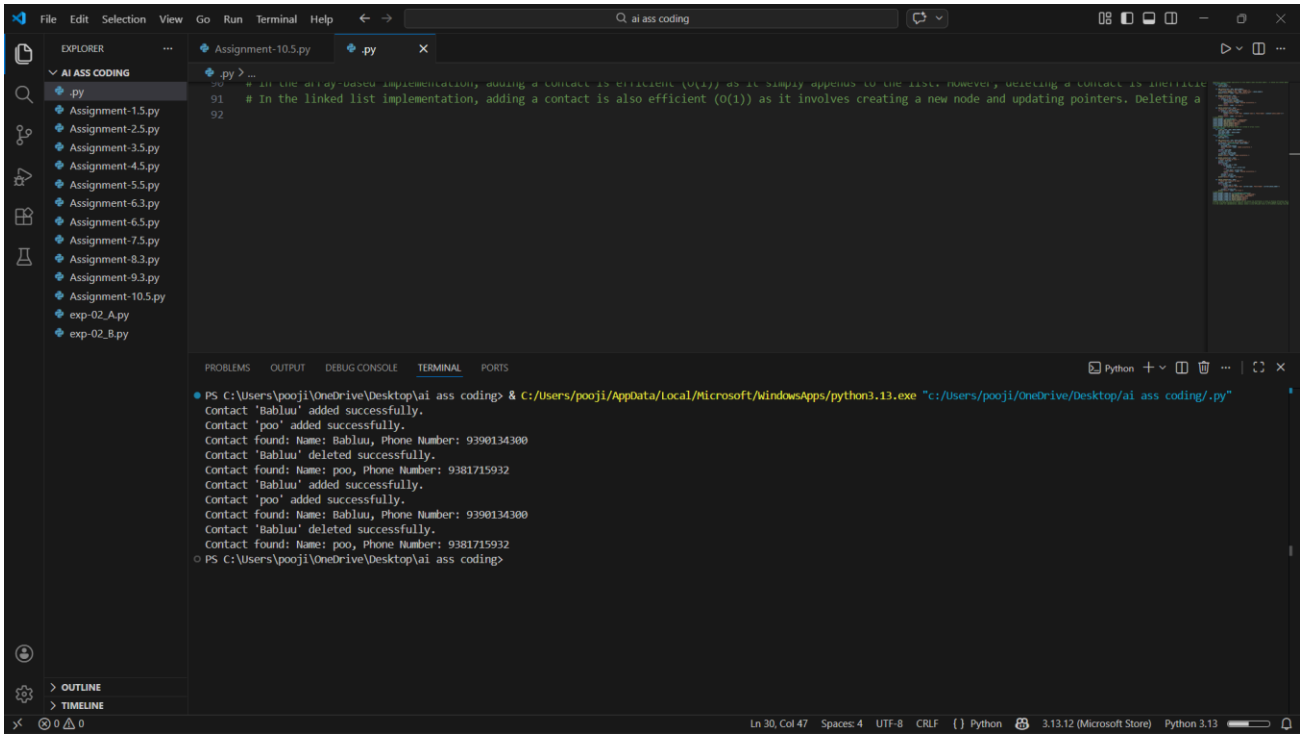




```
        self.next = None
class ContactManagerLinkedList:
    def __init__(self):
        self.head = None

    def add_contact(self, name, phone_number):
        """Add a new contact to the contact manager."""
        new_contact = ContactNode(name, phone_number)
        if not self.head:
            self.head = new_contact
            print(f"Contact '{name}' added successfully.")
            return
        current = self.head
        while current.next:
            current = current.next
        current.next = new_contact
        print(f"Contact '{name}' added successfully.")

    def delete_contact(self, name):
        """Delete a contact by name."""
        current = self.head
        previous = None
        while current:
            if current.name == name:
                if previous:
                    previous.next = current.next
                else:
                    self.head = current.next
                print(f"Contact '{name}' deleted successfully.")
                return
            previous = current
            current = current.next
        print(f"Contact '{name}' not found.")

    def search_contact(self, name):
        """Search for a contact by name."""
        current = self.head
        while current:
            if current.name == name:
```

```
class ContactManagerLinkedList:
    def delete_contact(self, name):
                if previous:
                    previous.next = current.next
                else:
                    self.head = current.next
                print(f"Contact '{name}' deleted successfully.")
                return
            previous = current
            current = current.next
        print(f"Contact '{name}' not found.")

    def search_contact(self, name):
        """Search for a contact by name."""
        current = self.head
        while current:
            if current.name == name:
                print(f"Contact found: Name: {current.name}, Phone Number: {current.phone_number}")
                return
            current = current.next
        print(f"Contact '{name}' not found.")
# Example usage
contact_manager_linked_list = ContactManagerLinkedList()
contact_manager_linked_list.add_contact("Babluu", "9390134300")
contact_manager_linked_list.add_contact("poo", "9381715932")
contact_manager_linked_list.search_contact("Babluu")
contact_manager_linked_list.delete_contact("Babluu")
contact_manager_linked_list.search_contact("poo")
#compare the above code with the previous one and analyze the differences in terms of deletion efficiency and insertion efficiency.
# In the array-based implementation, adding a contact is efficient (O(1)) as it simply appends to the list. However, deleting a contact is inefficie
# In the linked list implementation, adding a contact is also efficient (O(1)) as it involves creating a new node and updating pointers. Deleting a
```

Result:

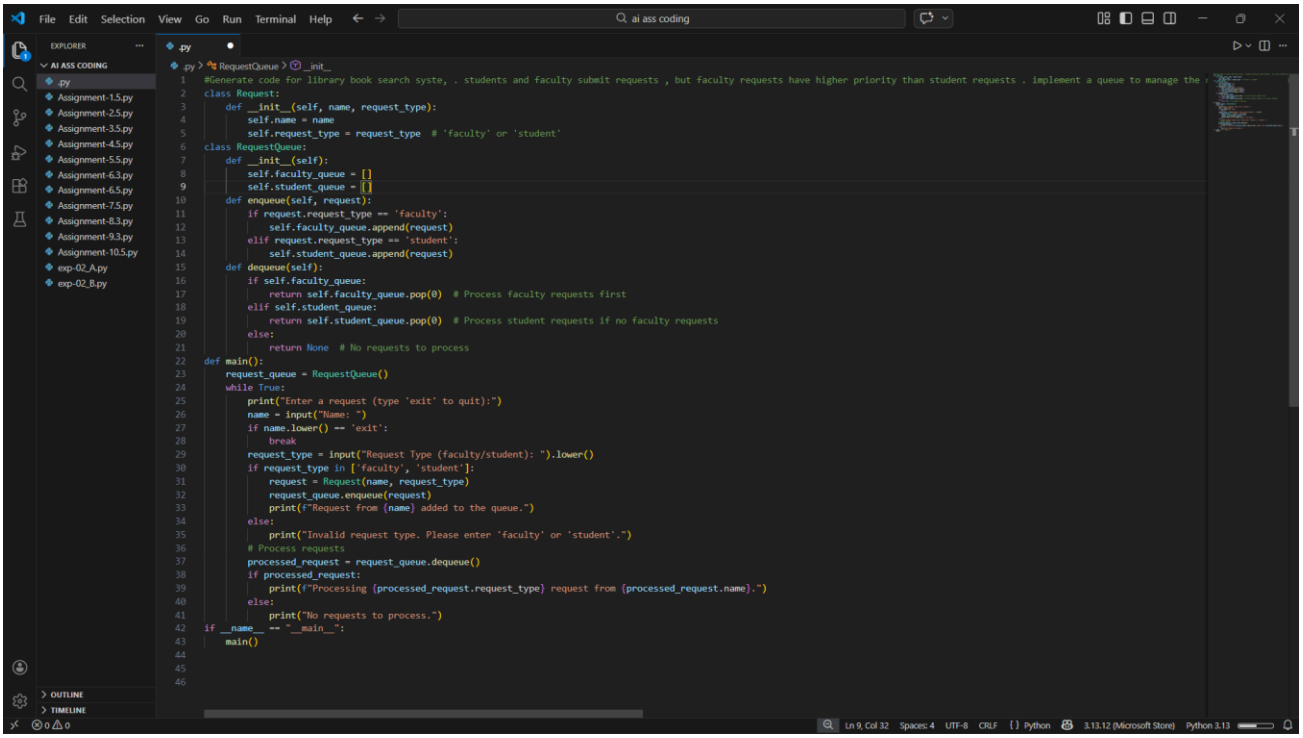


Observation:

- The array-based contact manager allows simple and fast appending (O(1)) but requires shifting elements during deletion, making removals O(n).

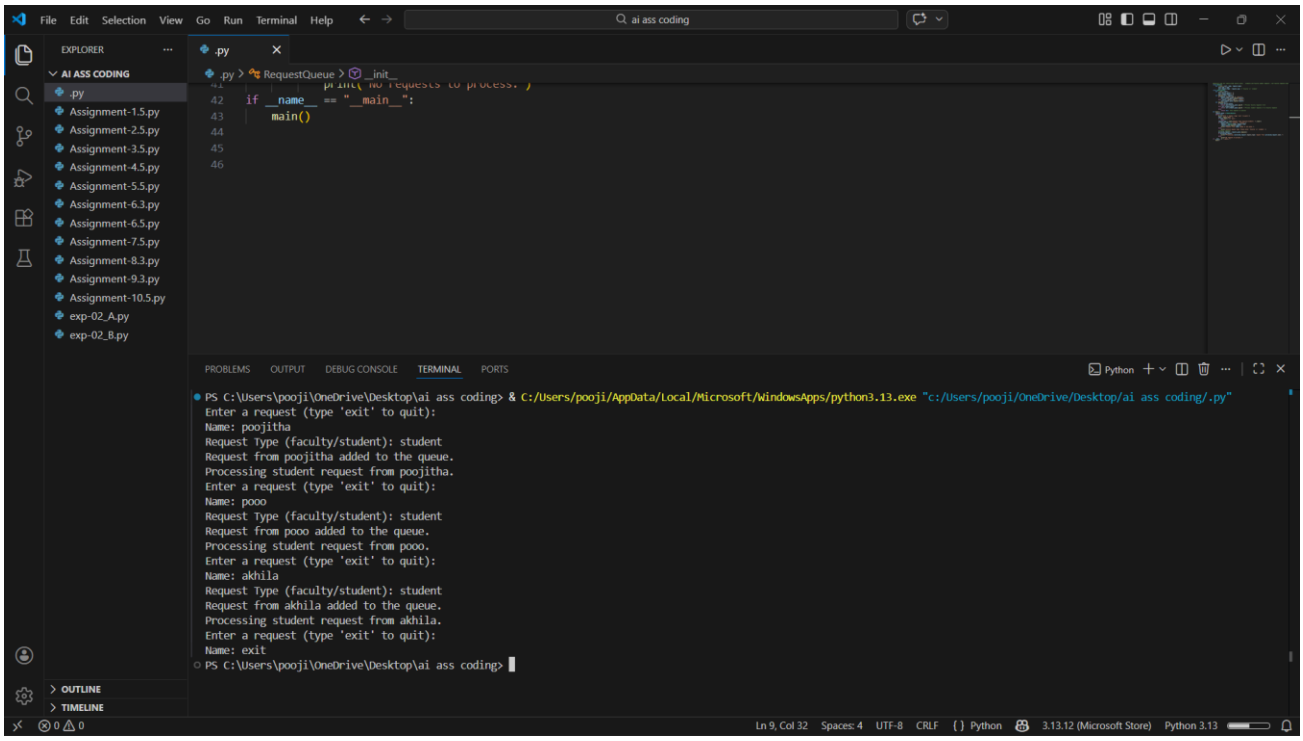
- The linked list version enables efficient insertion at the head (O(1)) without shifting on deletion, though searching remains O(n) in both approaches.

Task 2: Library Book Search System (Queues & Priority Queues)
Sample Input:
The SRU Library manages book borrow requests. Students and faculty submit requests, but faculty requests must be prioritized over student requests.
Tasks
1. Implement a Queue (FIFO) to manage book requests.
2. Extend the system to a Priority Queue, prioritizing faculty requests.
3. Use GitHub Copilot to assist in generating:
o enqueue() method
o dequeue() method
4. Test the system with a mix of student and faculty requests.
Prompt &Code:



```python
#Generate code for library book search syste, . students and faculty submit requests , but faculty requests have higher priority than student requests . implement a queue to manage the
class Request:
    def __init__(self, name, request_type):
        self.name = name
        self.request_type = request_type  # 'faculty' or 'student'
class RequestQueue:
    def __init__(self):
        self.faculty_queue = []
        self.student_queue = []
    def enqueue(self, request):
        if request.request_type == 'faculty':
            self.faculty_queue.append(request)
        elif request.request_type == 'student':
            self.student_queue.append(request)
    def dequeue(self):
        if self.faculty_queue:
            return self.faculty_queue.pop(0)  # Process faculty requests first
        elif self.student_queue:
            return self.student_queue.pop(0)  # Process student requests if no faculty requests
        else:
            return None  # No requests to process
def main():
    request_queue = RequestQueue()
    while True:
        print("Enter a request (type 'exit' to quit):")
        name = input("Name: ")
        if name.lower() == 'exit':
            break
        request_type = input("Request Type (faculty/student): ").lower()
        if request_type in ['faculty', 'student']:
            request = Request(name, request_type)
            request_queue.enqueue(request)
            print(f"Request from {name} added to the queue.")
        else:
            print("Invalid request type. Please enter 'faculty' or 'student'.")
        # Process requests
        processed_request = request_queue.dequeue()
        if processed_request:
            print(f"Processing {processed_request.request_type} request from {processed_request.name}.")
        else:
            print("No requests to process.")
if __name__ == "__main__":
    main()
```

Result:



```
PS C:\Users\pooji\OneDrive\Desktop\ai ass coding> & C:/Users/pooji/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/pooji/OneDrive/Desktop/ai ass coding/.py"
Enter a request (type 'exit' to quit):
Name: poojitha
Request Type (faculty/student): student
Request from poojitha added to the queue.
Processing student request from poojitha.
Enter a request (type 'exit' to quit):
Name: pooo
Request Type (faculty/student): student
Request from pooo added to the queue.
Processing student request from pooo.
Enter a request (type 'exit' to quit):
Name: akhila
Request Type (faculty/student): student
Request from akhila added to the queue.
Processing student request from akhila.
Enter a request (type 'exit' to quit):
Name: exit
PS C:\Users\pooji\OneDrive\Desktop\ai ass coding>
```

Observation:

- The priority queue ensures faculty requests are always processed before student requests, regardless of arrival order.

•Within each category (faculty or student), requests are handled in FIFO order, preserving fairness among the same requester type

•

Task 3: Emergency Help Desk (Stack Implementation)

SR University's IT Help Desk receives technical support tickets from students and staff. While tickets are received sequentially, issue escalation follows a Last-In, First-Out (LIFO) approach.

Tasks

1. Implement a Stack to manage support tickets.

2. Provide the following operations:

o push(ticket)

o pop()

o peek()

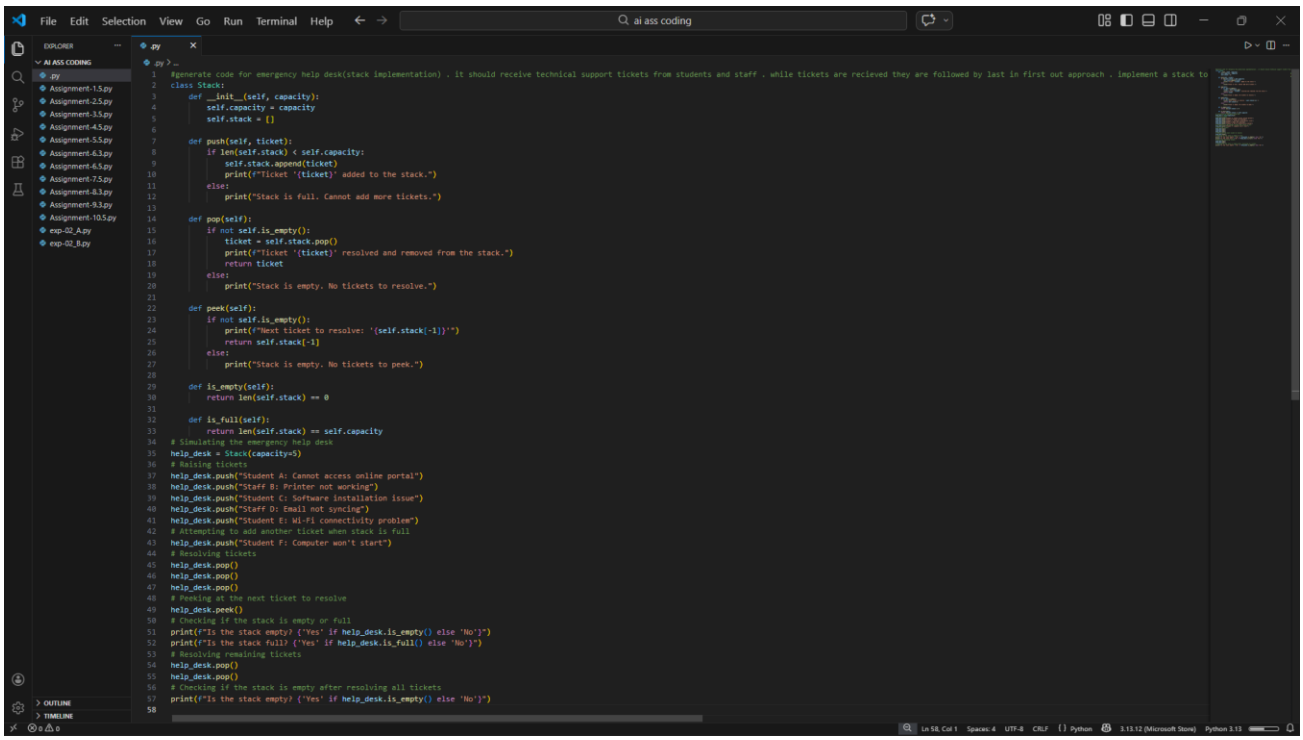
3. Simulate at least five tickets being raised and resolved.

4. Use GitHub Copilot to suggest additional stack operations such as:

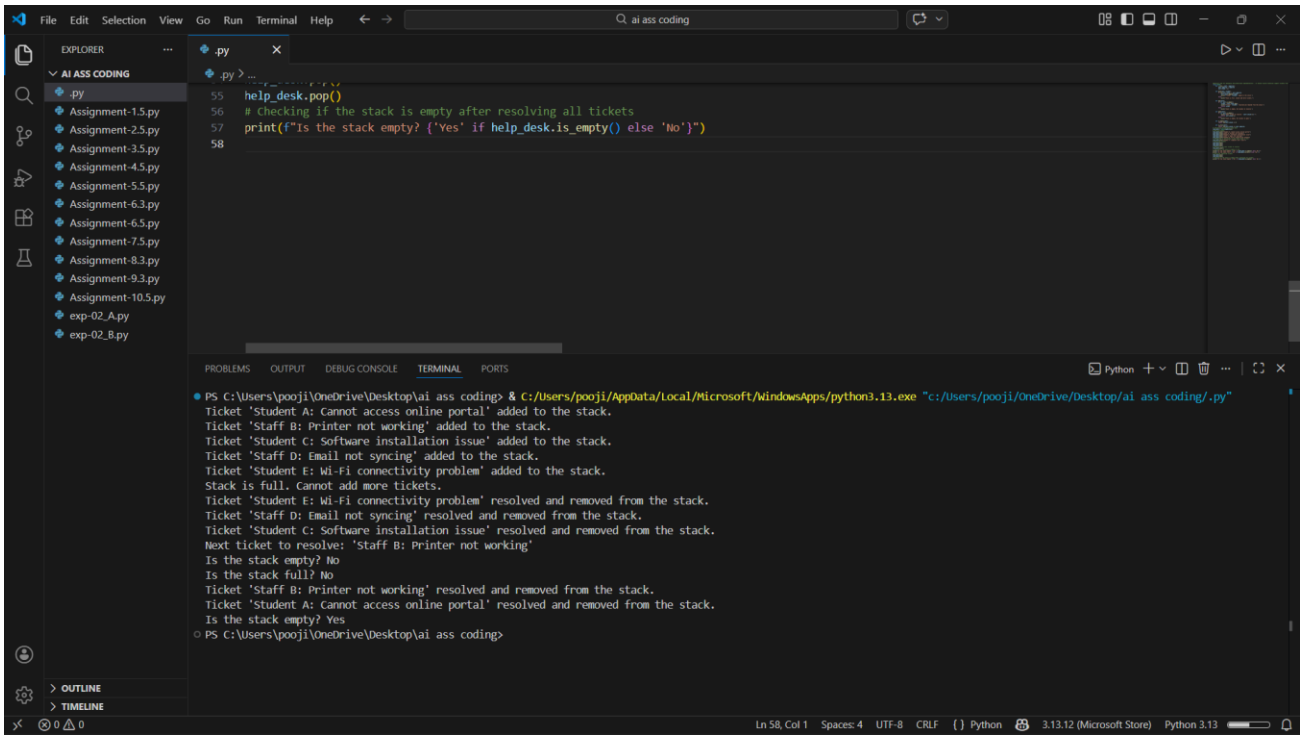
o Checking whether the stack is empty

o Checking whether the stack is full (if applicable)

Prompt & Code:



Result:



Observation:

The stack follows the LIFO (Last In, First Out) principle, so the most recently added support ticket is processed first.

The `peek()` method allows viewing the top ticket without removing it, and attempting to `pop()` from an empty stack safely returns `None`

Task 4: Hash Table

Objective

To implement a Hash Table and understand collision handling

Sample input:

Task Description

Use AI to generate a hash table with:

• Insert

• Search

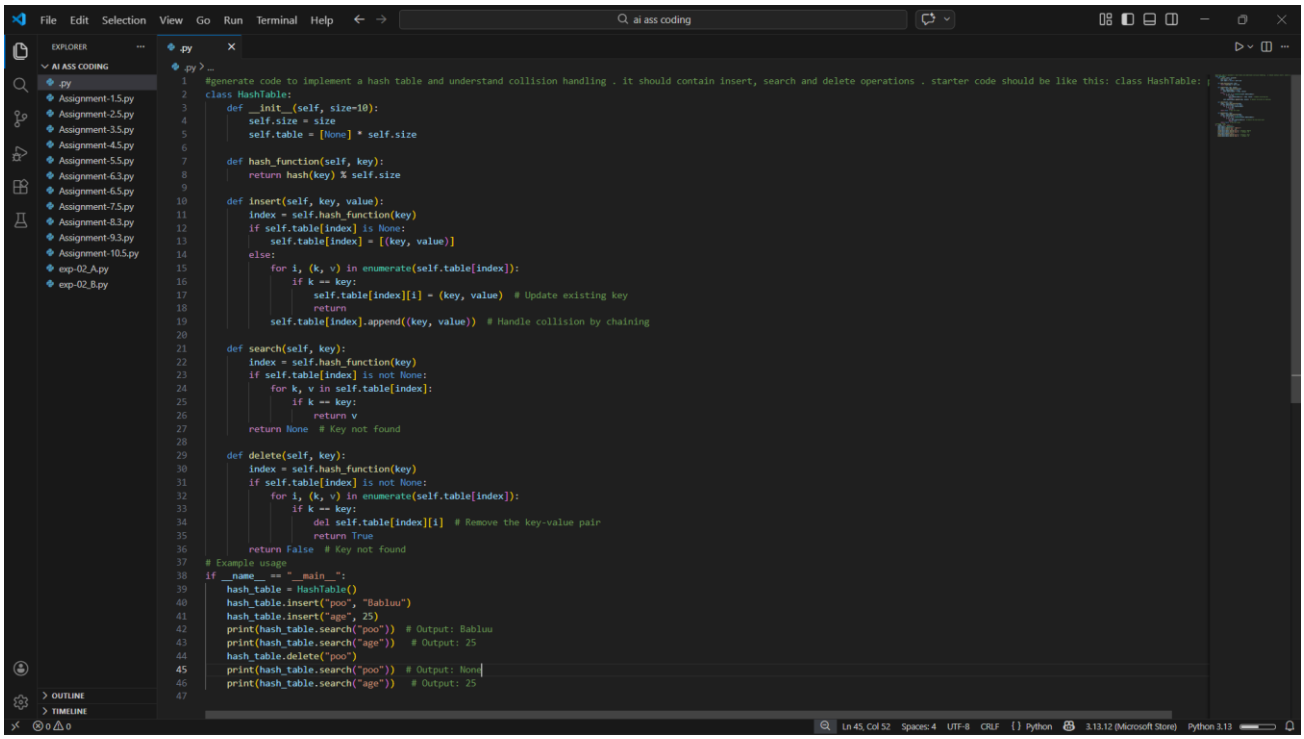
• Delete

Starter Code

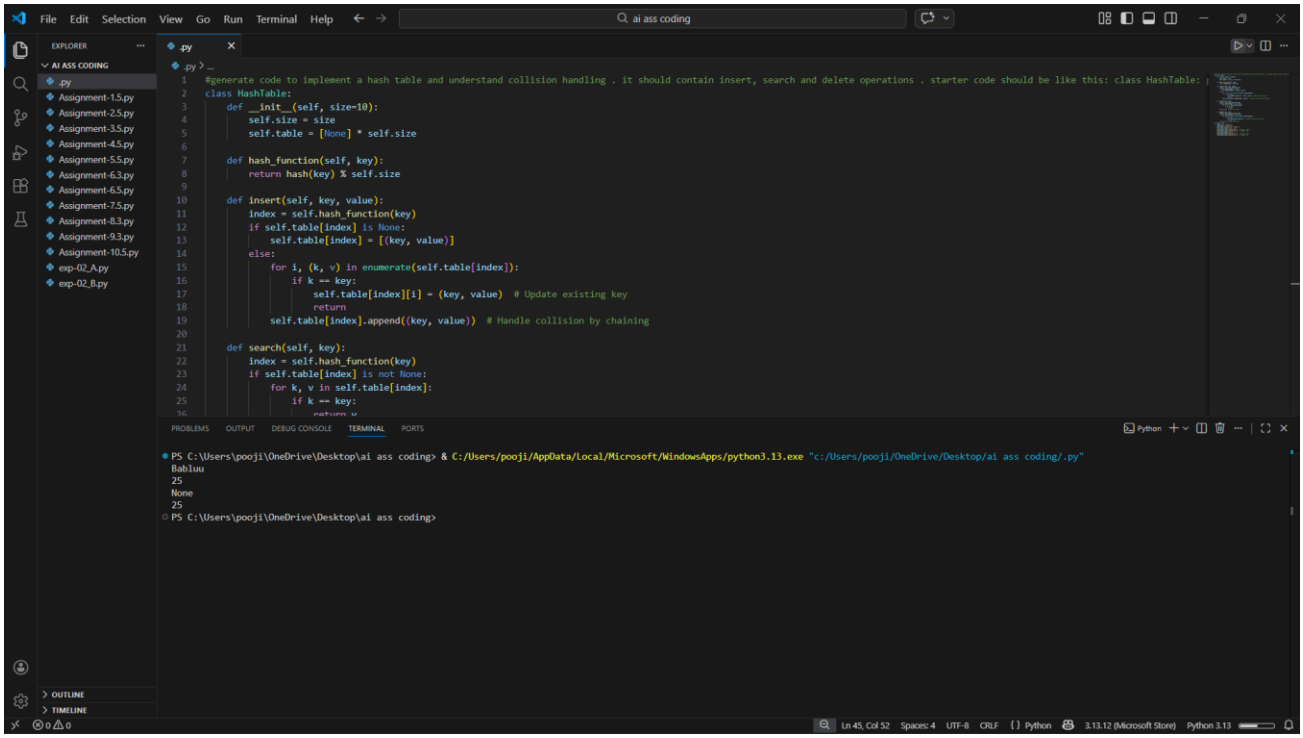
class HashTable:

pass

Prompt &Code:



Result:



Observation

The hash table provides fast average-case time complexity of O(1) for insertion, search, and deletion operations using key-based access.

After deleting a key, searching for it returns `None`, while other existing keys remain unaffected in the table.

Task 5: Real-Time Application Challenge

Sample input:

Design a Campus Resource Management System with the following features:

• Student Attendance Tracking

• Event Registration System

• Library Book Borrowing

• Bus Scheduling System

• Cafeteria Order Queue

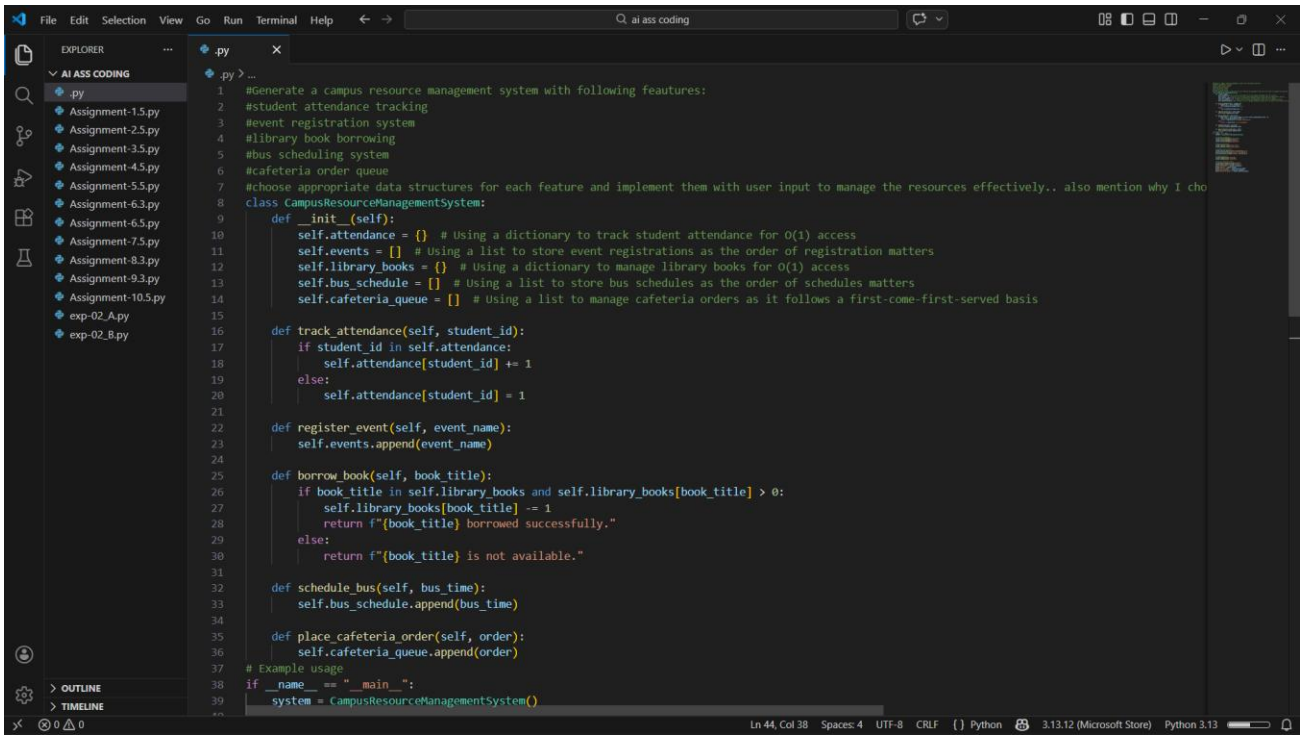
Student Tasks

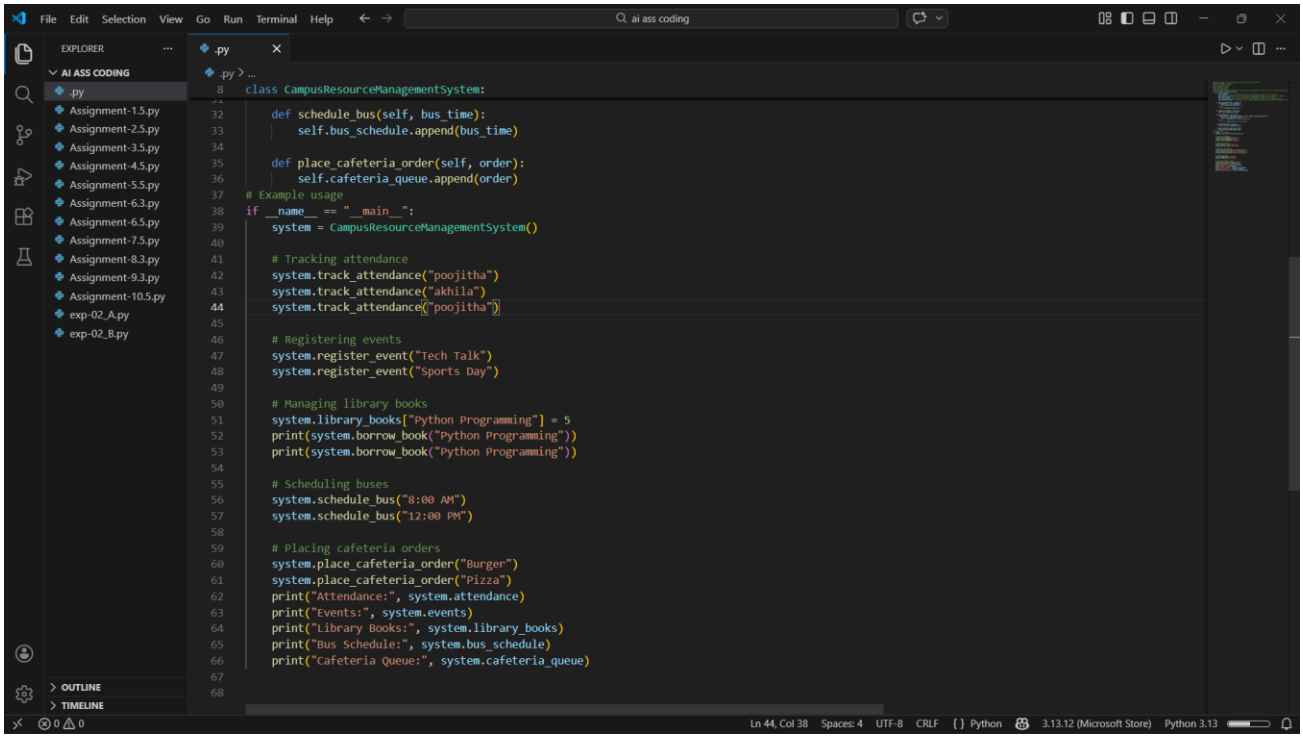
1. Choose the most appropriate data structure for each feature.

2. Justify your choice in 2–3 sentences.

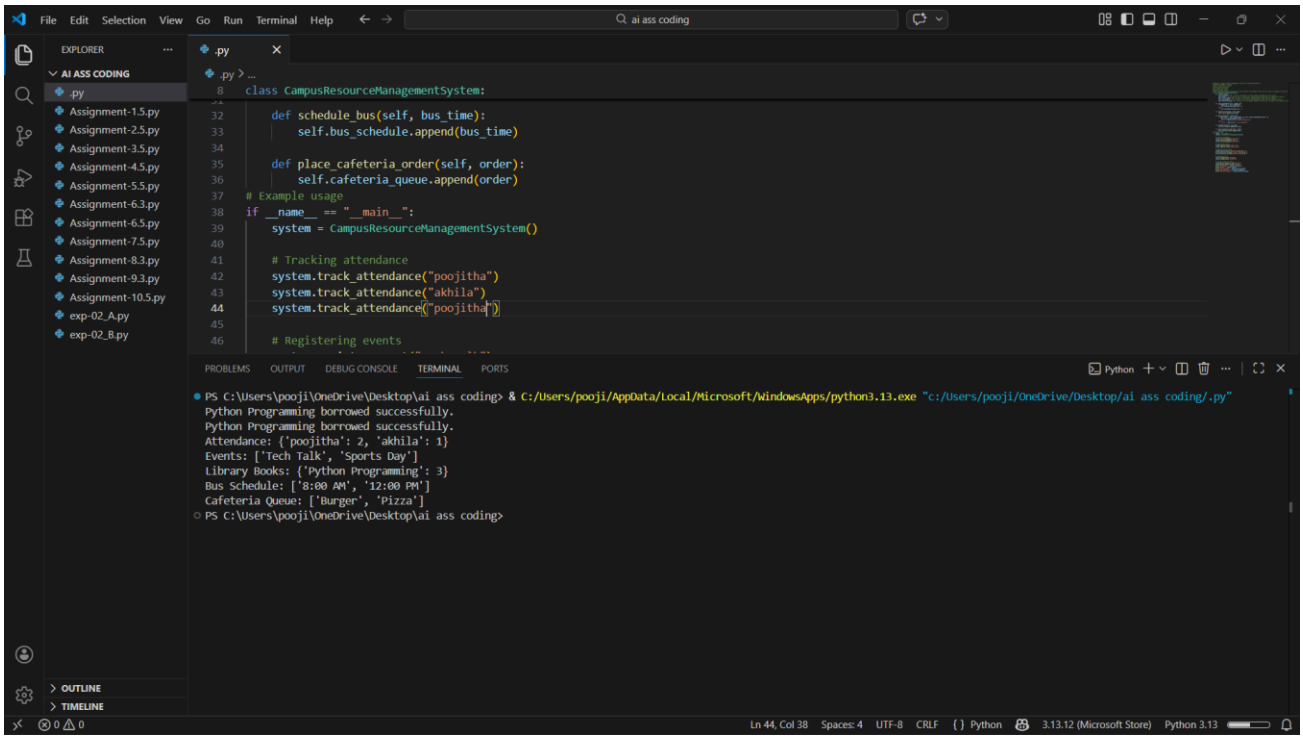
3. Implement one selected feature using AI-assisted code generation.

Prombt & Code:




File Edit Selection View Go Run Terminal Help
ai ass coding
EXPLORER
AI ASS CODING
.py
Assignment-1.5.py
Assignment-2.5.py
Assignment-3.5.py
Assignment-4.5.py
Assignment-5.5.py
Assignment-6.3.py
Assignment-6.5.py
Assignment-7.5.py
Assignment-8.3.py
Assignment-9.3.py
Assignment-10.5.py
exp-02_A.py
exp-02_B.py
OUTLINE
TIMELINE
class CampusResourceManagementSystem:
def schedule_bus(self, bus_time):
self.bus_schedule.append(bus_time)
def place_cafeteria_order(self, order):
self.cafeteria_queue.append(order)
# Example usage
if __name__ == "__main__":
system = CampusResourceManagementSystem()
# Tracking attendance
system.track_attendance("poojitha")
system.track_attendance("akhila")
system.track_attendance("poojitha")
# Registering events
system.register_event("Tech Talk")
system.register_event("Sports Day")
# Managing library books
system.library_books["Python Programming"] = 5
print(system.borrow_book("Python Programming"))
print(system.borrow_book("Python Programming"))
# Scheduling buses
system.schedule_bus("8:00 AM")
system.schedule_bus("12:00 PM")
# Placing cafeteria orders
system.place_cafeteria_order("Burger")
system.place_cafeteria_order("Pizza")
print("Attendance:", system.attendance)
print("Events:", system.events)
print("Library Books:", system.library_books)
print("Bus Schedule:", system.bus_schedule)
print("Cafeteria Queue:", system.cafeteria_queue)
Ln 44, Col 38 Spaces: 4 UTF-8 CRLF { } Python 3.13.12 (Microsoft Store) Python 3.13

Result:


File Edit Selection View Go Run Terminal Help
ai ass coding
EXPLORER
AI ASS CODING
.py
Assignment-1.5.py
Assignment-2.5.py
Assignment-3.5.py
Assignment-4.5.py
Assignment-5.5.py
Assignment-6.3.py
Assignment-6.5.py
Assignment-7.5.py
Assignment-8.3.py
Assignment-9.3.py
Assignment-10.5.py
exp-02_A.py
exp-02_B.py
OUTLINE
TIMELINE
class CampusResourceManagementSystem:
def schedule_bus(self, bus_time):
self.bus_schedule.append(bus_time)
def place_cafeteria_order(self, order):
self.cafeteria_queue.append(order)
# Example usage
if __name__ == "__main__":
system = CampusResourceManagementSystem()
# Tracking attendance
system.track_attendance("poojitha")
system.track_attendance("akhila")
system.track_attendance("poojitha")
# Registering events
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python
PS C:\Users\pooji\OneDrive\Desktop\ai ass coding> & C:/Users/pooji/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/pooji/OneDrive/Desktop/ai ass coding/.py"
Python Programming borrowed successfully.
Python Programming borrowed successfully.
Attendance: {'poojitha': 2, 'akhila': 1}
Events: ['Tech Talk', 'Sports Day']
Library Books: {'Python Programming': 3}
Bus Schedule: ['8:00 AM', '12:00 PM']
Cafeteria Queue: ['Burger', 'Pizza']
PS C:\Users\pooji\OneDrive\Desktop\ai ass coding>
Ln 44, Col 38 Spaces: 4 UTF-8 CRLF { } Python 3.13.12 (Microsoft Store) Python 3.13

Observation:

The dictionary efficiently tracks attendance using student IDs as keys, allowing quick O(1) access to each student's status.

The attendance report is generated by counting boolean values, making it simple to calculate total, present, and absent students dynamically.