

ASSIGNMENT- 11.3

Name:E.POOJITHA

HT.No: 2303A51356

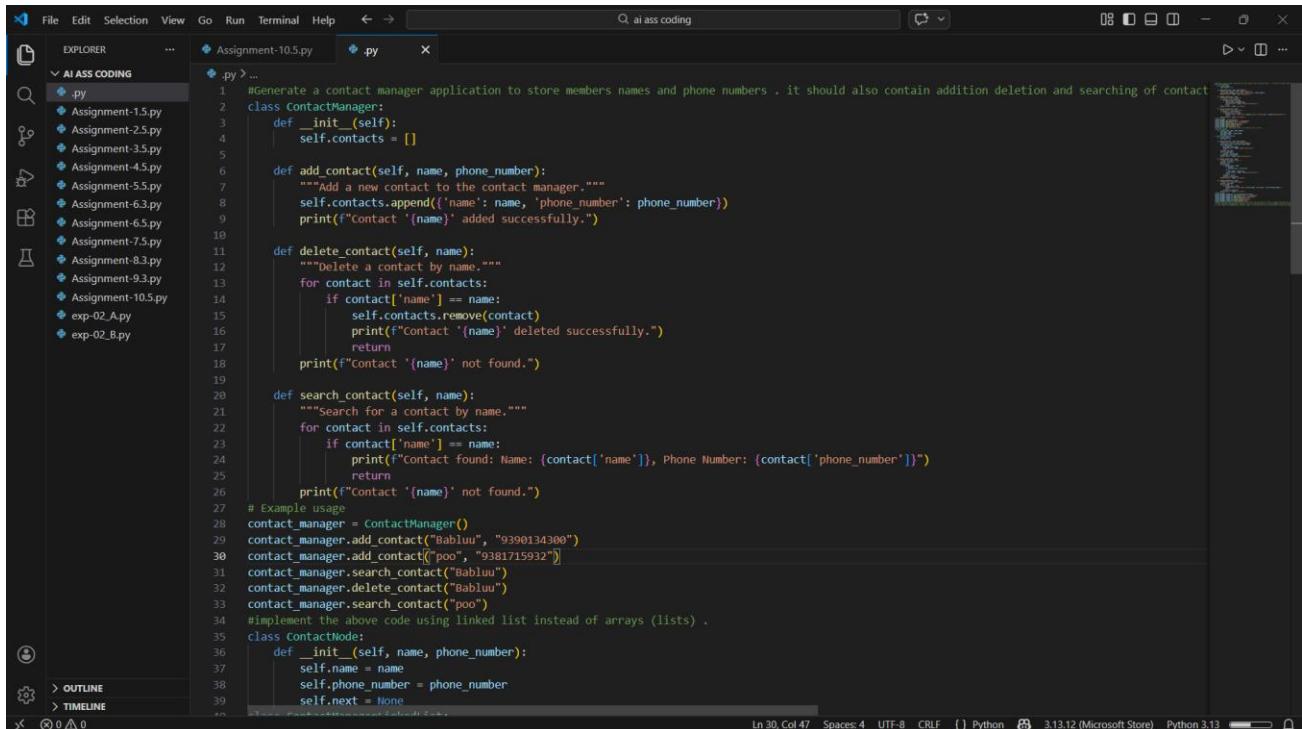
Batch: 20

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



The screenshot shows a Microsoft Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Search Bar:** Q ai ass coding.
- Editor Area:** The code for "Assignment-10.5.py" is displayed. The code implements a ContactManager class using lists to store contacts and provides methods for adding, deleting, and searching contacts.
- Left Sidebar:** Explorer view showing files like 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, and exp-02_B.py.
- Bottom Status Bar:** Lines 30, Col 47, Spaces: 4, UTF-8, CRLF, Python 3.13.12 (Microsoft Store), Python 3.13.

```
#Generate a contact manager application to store members names and phone numbers . it should also contain addition deletion and searching of contact
class ContactManager:
    def __init__(self):
        self.contacts = []

    def add_contact(self, name, phone_number):
        """Add a new contact to the contact manager."""
        self.contacts.append({'name': name, 'phone_number': phone_number})
        print(f"Contact '{name}' added successfully.")

    def delete_contact(self, name):
        """Delete a contact by name."""
        for contact in self.contacts:
            if contact['name'] == name:
                self.contacts.remove(contact)
                print(f"Contact '{name}' deleted successfully.")
                return
        print(f"Contact '{name}' not found.")

    def search_contact(self, name):
        """Search for a contact by name."""
        for contact in self.contacts:
            if contact['name'] == name:
                print(f"Contact found: Name: {contact['name']}, Phone Number: {contact['phone_number']}")  

                return
        print(f"Contact '{name}' not found.")

# Example usage
contact_manager = ContactManager()
contact_manager.add_contact("Babliu", "9390134300")
contact_manager.add_contact("poo", "9381715932")
contact_manager.search_contact("Babliu")
contact_manager.delete_contact("Babliu")
contact_manager.delete_contact("poo")
#implement the above code using linked list instead of arrays (lists).
class ContactNode:
    def __init__(self, name, phone_number):
        self.name = name
        self.phone_number = phone_number
        self.next = None
```

```

File Edit Selection View Go Run Terminal Help < > ai ass coding
EXPLORER .py Assignment-1.5.py Assignment-2.5.py Assignment-3.5.py Assignment-4.5.py Assignment-5.5.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
Assignment-10.5.py .py
35     class ContactNode:
36         self.next = None
37
38     class ContactManagerLinkedList:
39         def __init__(self):
40             self.head = None
41
42         def add_contact(self, name, phone_number):
43             """Add a new contact to the contact manager."""
44             new_contact = ContactNode(name, phone_number)
45             if not self.head:
46                 self.head = new_contact
47                 print(f"Contact '{name}' added successfully.")
48                 return
49             current = self.head
50             while current.next:
51                 current = current.next
52             current.next = new_contact
53             print(f"Contact '{name}' added successfully.")
54
55         def delete_contact(self, name):
56             """Delete a contact by name."""
57             current = self.head
58             previous = None
59             while current:
60                 if current.name == name:
61                     if previous:
62                         previous.next = current.next
63                     else:
64                         self.head = current.next
65                     print(f"Contact '{name}' deleted successfully.")
66                     return
67                 previous = current
68                 current = current.next
69             print(f"Contact '{name}' not found.")
70
71         def search_contact(self, name):
72             """Search for a contact by name."""
73             current = self.head
74             while current:
75                 if current.name == name:
76                     print(f"Contact found: Name: {current.name}, Phone Number: {current.phone_number}")
77                     return
78                 current = current.next
79             print(f"Contact '{name}' not found.")
80
81 # Example usage
82 contact_manager_linked_list = ContactManagerLinkedList()
83 contact_manager_linked_list.add_contact("Babluu", "9900134300")
84 contact_manager_linked_list.add_contact("poo", "9381715932")
85 contact_manager_linked list.search_contact("Babluu")
86 contact_manager_linked list.delete_contact("Babluu")
87 contact_manager_linked list.add_contact("Babluu")
88 contact_manager_linked list.search_contact("poo")
89 #Compare the above code with the previous one and analyze the differences in terms of deletion efficiency and insertion efficiency.
90 # In the array-based implementation, adding a contact is efficient ( $O(1)$ ) as it simply appends to the list. However, deleting a contact is inefficient.
91 # 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
92

```

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"
contact 'Babluu' added successfully.
contact 'poo' added successfully.
Contact found: Name: Babluu, Phone Number: 9390134300
contact 'Babluu' deleted successfully.
contact found: Name: poo, Phone Number: 9381715932
contact 'Babluu' added successfully.
contact 'poo' added successfully.
contact found: Name: Babluu, Phone Number: 9390134300
contact 'Babluu' deleted successfully.
contact found: Name: poo, Phone Number: 9381715932
PS C:\Users\pooji\OneDrive\Desktop\ai ass coding>

```

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:

The screenshot shows the Microsoft Visual Studio Code interface with the following details:

- File Explorer:** Shows a folder named "AI ASS CODING" containing several Python files: Assignment-1.5.py, Assignment-2.5.py, Assignment-3.5.py, Assignment-4.5.py, Assignment-5.5.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, and exp-02_B.py.
- Code Editor:** Displays the content of "RequestQueue.py". The code implements a queue to manage requests from students and faculty. It includes classes for Request and RequestQueue, and a main function that processes user input to add requests to the queue and print them out.
- Status Bar:** Shows the file path as "C:/Users/pooji/Desktop/ai ass coding/.py", line 9, column 32, and other status information like "Spaces: 4", "UTF-8", "Python 3.13", and "3.13.12 (Microsoft Store)".

Result:

The screenshot shows the Microsoft Visual Studio Code interface with the following details:

- File Explorer:** Same as the previous screenshot, showing the "AI ASS CODING" folder.
- Code Editor:** Displays the same "RequestQueue.py" code as above.
- Terminal:** Shows the output of running the script:

```
PS C:\Users\pooji\Desktop\ai ass coding & C:/Users/pooji/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/pooji/Desktop/ai ass coding/.py"
Enter a request (type 'exit' to quit):
Name: pooja
Request Type (faculty/student): student
Request from pooja added to the queue.
Processing student request from pooja.
Enter a request (type 'exit' to quit):
Name: poo
Request Type (faculty/student): student
Request from poo added to the queue.
Processing student request from poo.
Enter a request (type 'exit' to quit):
Name: akila
Request Type (faculty/student): student
Request from akila added to the queue.
Processing student request from akila.
Enter a request (type 'exit' to quit):
Name: exit
```
- Status Bar:** Shows the file path as "C:/Users/pooji/Desktop/ai ass coding/.py", line 9, column 32, and other status information like "Spaces: 4", "UTF-8", "Python 3.13", and "3.13.12 (Microsoft Store)".

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:

The screenshot shows a Microsoft Visual Studio Code interface with the following details:

- File Explorer:** Shows a folder named "AI ASS CODING" containing several Python files: 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_App, and exp-02_B.py.
- Code Editor:** Displays a Python script named ".py". The code defines a class `help_desk` with a method `pop()`. It then checks if the stack is empty after resolving all tickets and prints a message based on the result.

```
55     help_desk.pop()
56     # Checking if the stack is empty after resolving all tickets
57     print(f"Is the stack empty? {'Yes' if help_desk.is_empty() else 'No'})")
58
```
- Terminal:** Shows the output of running the script. The terminal window title is "Python". The output displays a series of ticket entries being added to a stack and then resolved, followed by a check for stack emptiness.

```
PS C:\Users\pooji\OneDrive\Desktop\ai ass coding & C:/Users/pooji/AppData/Local/Microsoft/Windows/Apps/python3.13.exe "c:/Users/pooji/OneDrive/Desktop/ai ass coding/.py"
Ticket 'Student A: Cannot access online portal' added to the stack.
Ticket 'Staff B: Printer not working' added to the stack.
Ticket 'Student C: Software installation issue' added to the stack.
Ticket 'Staff D: Email not syncing' added to the stack.
Ticket 'Student E: Wi-Fi connectivity problem' added to the stack.
Stack is full. Cannot add more tickets.
Ticket 'Student E: Wi-Fi connectivity problem' resolved and removed from the stack.
Ticket 'Staff D: Email not syncing' resolved and removed from the stack.
Ticket 'Student C: Software installation issue' resolved and removed from the stack.
Next ticket to resolve: 'Staff B: Printer not working'.
Is the stack empty? No
Is the stack full? No
Ticket 'Staff B: Printer not working' resolved and removed from the stack.
Ticket 'Student A: Cannot access online portal' resolved and removed from the stack.
Is the stack empty? Yes
PS C:\Users\pooji\OneDrive\Desktop\ai ass coding>
```
- Sidebar:** Includes sections for "OUTLINE" and "TIMELINE".

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:

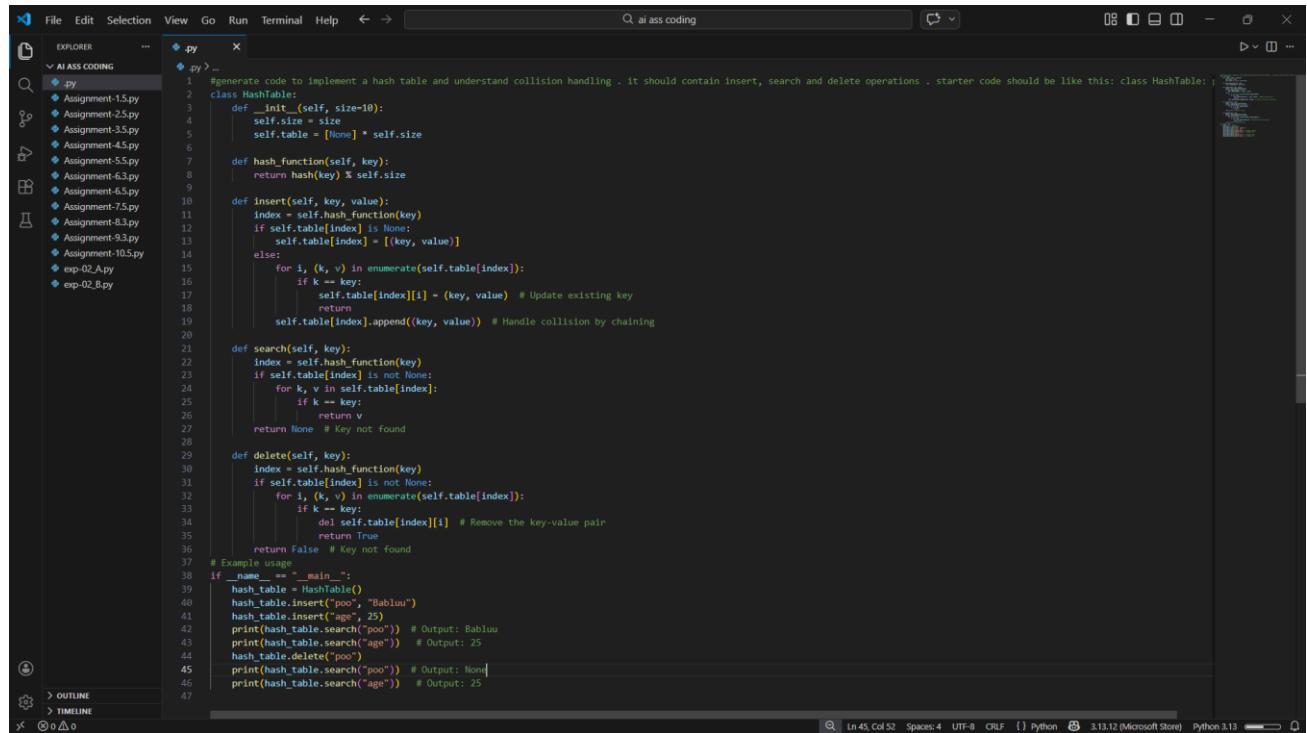
- Insert
- Search
- Delete

Starter Code

class HashTable:

pass

Prompt &Code:



The screenshot shows the Microsoft Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Search Bar:** ai ass coding.
- Explorer:** Shows a folder named "AI ASS CODING" containing several Python files (Assignment-1.5.py, Assignment-2.5.py, Assignment-3.5.py, Assignment-4.5.py, Assignment-5.5.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) and a ".py" file.
- Code Editor:** Displays the generated code for a HashTable class. The code includes methods for insert, search, and delete operations, using chaining for collisions. It also includes an example usage section at the bottom.
- Status Bar:** L1 45, Col 52, Spaces: 4, UTF-8, CR LF, Python 3.13.12 (Microsoft Store), Python 3.13.

Result:

```
#generate code to implement a hash table and understand collision handling . it should contain insert, search and delete operations . starter code should be like this: class HashTable:
    def __init__(self, size=10):
        self.size = size
        self.table = [None] * self.size

    def hash_function(self, key):
        return hash(key) % self.size

    def insert(self, key, value):
        index = self.hash_function(key)
        if self.table[index] is None:
            self.table[index] = [(key, value)]
        else:
            for i, (k, v) in enumerate(self.table[index]):
                if k == key:
                    self.table[index][i] = (key, value) # Update existing key
                    return
            self.table[index].append((key, value)) # Handle collision by chaining

    def search(self, key):
        index = self.hash_function(key)
        if self.table[index] is not None:
            for k, v in self.table[index]:
                if k == key:
                    return v
        return None
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\pooji\Desktop\ai ass coding> & C:/Users/pooji/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/pooji/Desktop/ai ass coding/.py"

Bablu

25

None

25

PS C:\Users\pooji\Desktop\ai ass coding>

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.

Prompt & Code:

The screenshot shows a Microsoft Visual Studio Code interface with the following details:

- File Explorer:** Shows a folder named "AI ASS CODING" containing several Python files: .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, and exp-02_B.py.
- Code Editor:** Displays a Python script (.py) with the following code:

```
#Generate a campus resource management system with following feautures:
#student attendance tracking
#event registration system
#library book borrowing
#bus scheduling system
#cafeteria order queue
#choose appropriate data structures for each feature and implement them with user input to manage the resources effectively.. also mention why I cho
class CampusResourceManagementSystem:
    def __init__(self):
        self.attendance = {} # Using a dictionary to track student attendance for O(1) access
        self.events = [] # Using a list to store event registrations as the order of registration matters
        self.library_books = {} # Using a dictionary to manage library books for O(1) access
        self.bus_schedule = [] # Using a list to store bus schedules as the order of schedules matters
        self.cafeteria_queue = [] # Using a list to manage cafeteria orders as it follows a first-come-first-served basis

    def track_attendance(self, student_id):
        if student_id in self.attendance:
            self.attendance[student_id] += 1
        else:
            self.attendance[student_id] = 1

    def register_event(self, event_name):
        self.events.append(event_name)

    def borrow_book(self, book_title):
        if book_title in self.library_books and self.library_books[book_title] > 0:
            self.library_books[book_title] -= 1
            return f"{book_title} borrowed successfully."
        else:
            return f"{book_title} is not available."

    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()
```

The status bar at the bottom indicates: Ln 44, Col 38 Spaces: 4 UTF-8 CRLF () Python 3.13.12 (Microsoft Store) Python 3.13

The screenshot shows the Microsoft Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows a folder named "AI ASS CODING" containing several Python files: 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, and exp-02_B.py.
- Code Editor (Center):** Displays the content of the file "ai ass coding.py". The code defines a class `CampusResourceManagementSystem` with methods for scheduling buses, placing cafeteria orders, tracking attendance, registering events, managing library books, and more. It includes example usage at the bottom.
- Bottom Status Bar:** Shows the current line (Ln 44), column (Col 38), spaces used (Spaces: 4), encoding (UTF-8), CRLF, Python version (Python 3.13), and the Microsoft Store link.

Result:

The screenshot shows the Microsoft Visual Studio Code interface with the following details:

- File Explorer (Left):** Same as the previous screenshot, showing the "AI ASS CODING" folder and its contents.
- Terminal (Bottom):** Displays the command-line output of running the script:

```
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']
```
- Bottom Status Bar:** Shows the current line (Ln 44), column (Col 38), spaces used (Spaces: 4), encoding (UTF-8), CRLF, Python version (Python 3.13), and the Microsoft Store link.

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.