

## ASSIGNMENT-11.3

Name: E.Ramya

Ht.no: 2303A51282

Batch: 05

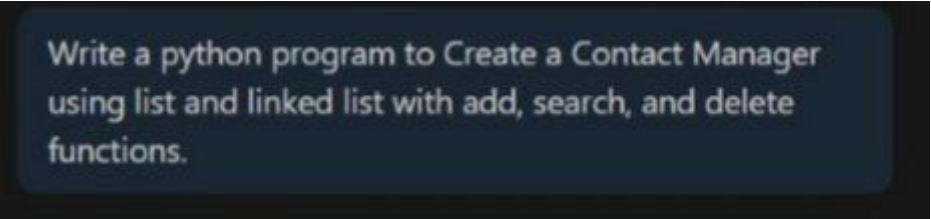
### **Task 1:** Smart Contact Manager (Arrays & Linked Lists)

#### Scenario

SR University's student club requires a simple Contact Manager Application to store members' names and phone numbers. The system should support efficient addition, searching, and deletion of contacts.

#### **Prompt:**

Write a python program to Create a Contact Manager using list and linked list with add, search, and delete functions.



Write a python program to Create a Contact Manager using list and linked list with add, search, and delete functions.

**Code:**

[illegible]

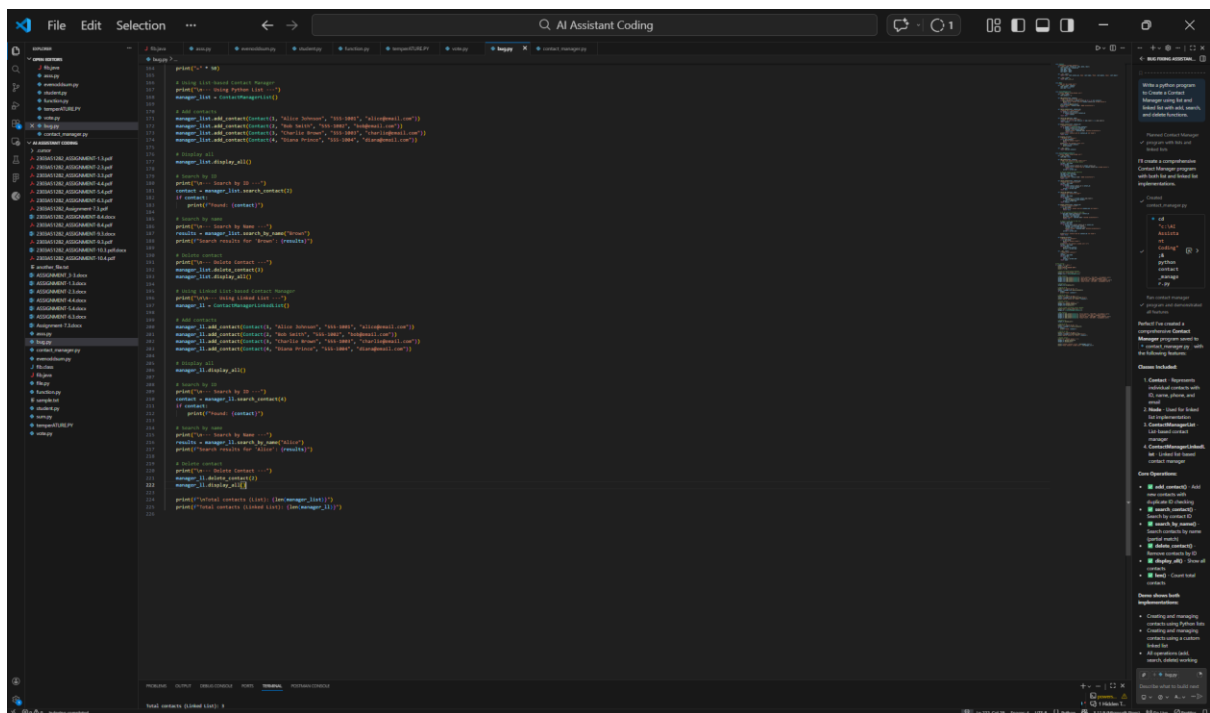
The screenshot displays the AI Assistant Coding environment. The main editor shows a C++ file named `contact_manager.cpp` with the following code:

```

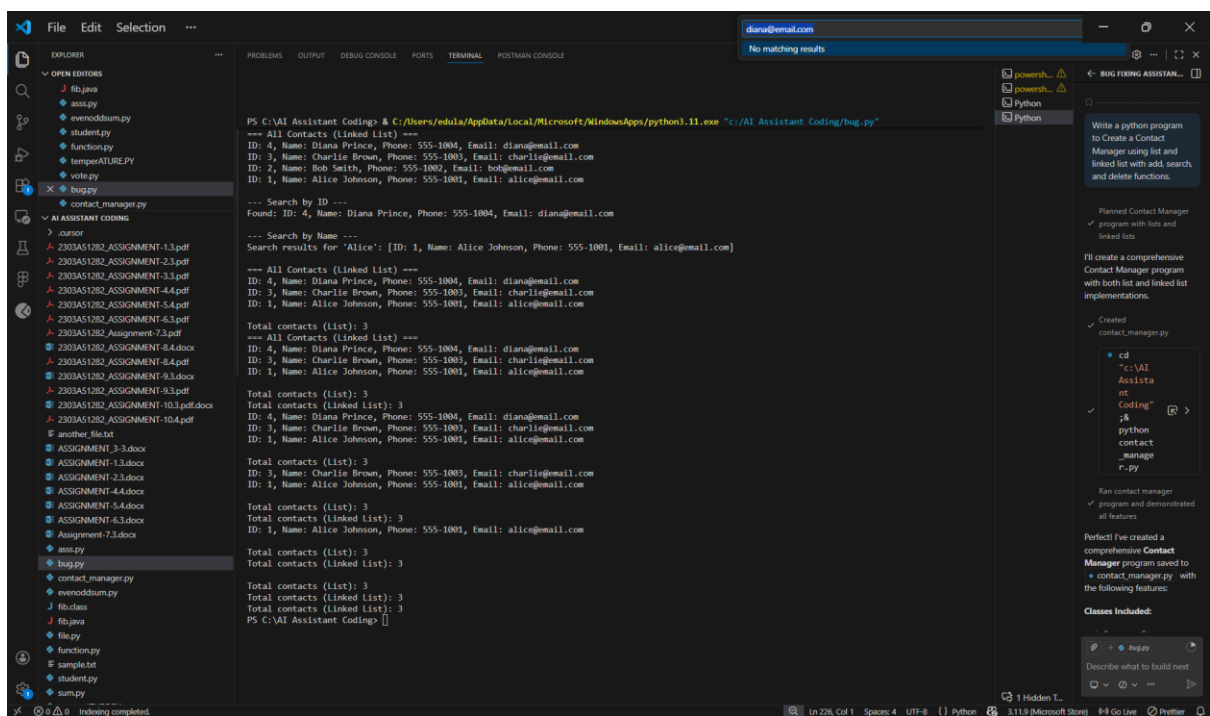
1 // contact_manager.cpp
2 #include <iostream>
3 #include <string>
4 #include <vector>
5 #include <map>
6 #include <algorithm>
7 using namespace std;
8
9 struct Contact {
10     string name;
11     string phone;
12     string email;
13 };
14
15 vector<Contact> contacts;
16
17 void add_contact(string name, string phone, string email) {
18     Contact c;
19     c.name = name;
20     c.phone = phone;
21     c.email = email;
22     contacts.push_back(c);
23 }
24
25 void delete_contact(string name) {
26     for (auto &contact : contacts) {
27         if (contact.name == name) {
28             contacts.erase(remove(contacts.begin(), contacts.end(), contact));
29         }
30     }
31 }
32
33 void search_by_name(string name) {
34     for (auto &contact : contacts) {
35         if (contact.name == name) {
36             cout << "Found contact: " << contact.name << endl;
37             cout << "Phone: " << contact.phone << endl;
38             cout << "Email: " << contact.email << endl;
39         }
40     }
41 }
42
43 void display_contacts() {
44     if (contacts.empty()) {
45         cout << "No contacts found." << endl;
46     } else {
47         for (auto &contact : contacts) {
48             cout << "Name: " << contact.name << ", Phone: " << contact.phone << ", Email: " << contact.email << endl;
49         }
50     }
51 }
52
53 int main() {
54     int choice;
55     do {
56         cout << "1. Add contact\n";
57         cout << "2. Delete contact\n";
58         cout << "3. Search by name\n";
59         cout << "4. Display all contacts\n";
60         cout << "5. Exit\n";
61         choice = 0;
62         while (choice < 1 || choice > 5) {
63             choice = 0;
64             cout << "Enter your choice: ";
65             for (int i = 1; i <= 5; i++) {
66                 cout << i << " ";
67                 if (i % 5 == 0) cout << endl;
68             }
69             cout << " ";
70             cin >> choice;
71         }
72         switch (choice) {
73             case 1: {
74                 string name, phone, email;
75                 cout << "Enter name: ";
76                 getline(cin, name);
77                 cout << "Enter phone: ";
78                 getline(cin, phone);
79                 cout << "Enter email: ";
80                 getline(cin, email);
81                 add_contact(name, phone, email);
82                 break;
83             }
84             case 2: {
85                 string name;
86                 cout << "Enter name to delete: ";
87                 getline(cin, name);
88                 delete_contact(name);
89                 break;
90             }
91             case 3: {
92                 string name;
93                 cout << "Enter name to search: ";
94                 getline(cin, name);
95                 search_by_name(name);
96                 break;
97             }
98             case 4: {
99                 display_contacts();
100                break;
101            }
102            case 5: {
103                return 0;
104            }
105        }
106    } while (choice != 5);
107    return 0;
108 }

```

The right-hand sidebar shows a 'Run' button and a 'Terminal' output area. The terminal output shows the program's execution, including the prompt 'Enter your choice: ' and the resulting output of the program.



## Output:



## Explanation:

- In an array, adding at the end is fast, but inserting in the middle is slow because elements must shift.
- In a linked list, insertion is fast because no shifting is needed.
- Searching takes the same time in both (you must check each element).
- Deleting in an array is slower due to shifting elements.
- Linked list is better for frequent insertions and deletions.

## **Task 2:** Library Book Search System (Queues & Priority Queues)

### Scenario

The SRU Library manages book borrow requests. Students and faculty submit requests, but faculty requests must be prioritized over student requests.

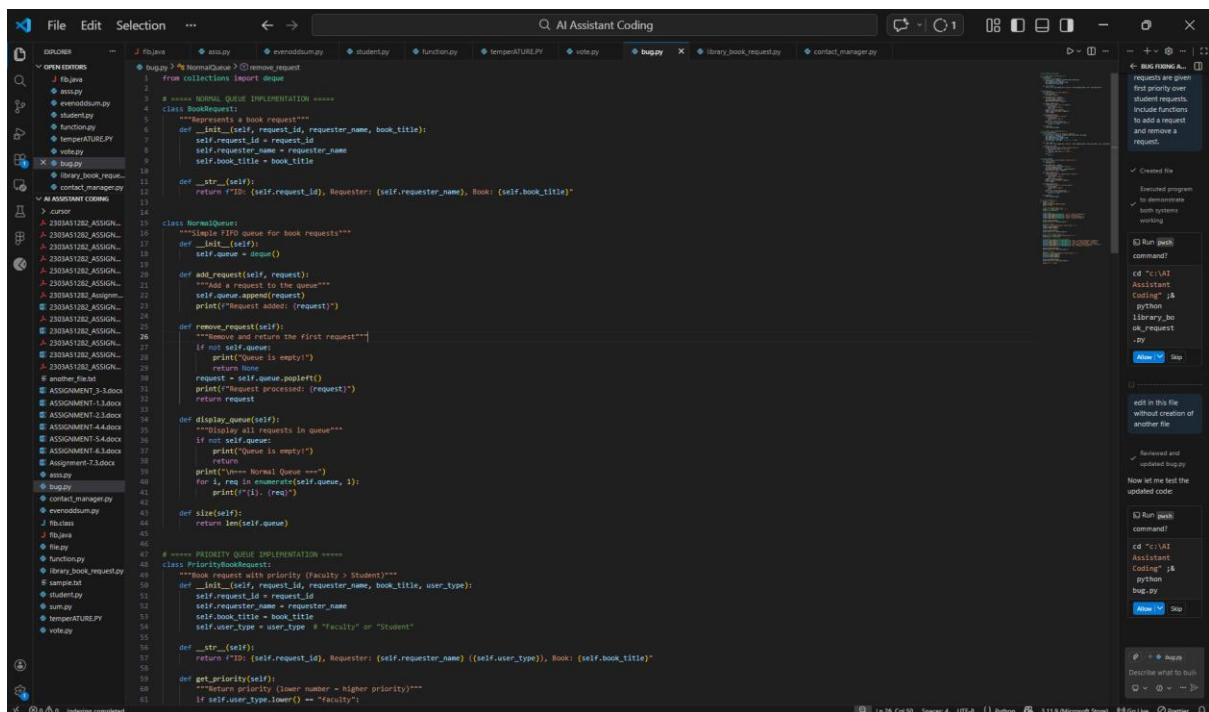
### **Prompt:**

Write a Python program for a library book request system. First, make a normal queue where requests are handled in the order they come. Then, make another version where faculty requests are given

first priority over student requests. Include functions to add a request and remove a request.

Write a Python program for a library book request system.  
First, make a normal queue where requests are handled in the order they come.  
Then, make another version where faculty requests are given first priority over student requests.  
Include functions to add a request and remove a request.

Code:



```
1 #buggy>%NonInteractive>@remove_request
2 from collections import deque
3
4 # ===== NORMAL QUEUE IMPLEMENTATION =====
5 class BookRequest:
6     """Represents a book request"""
7     def __init__(self, request_id, requester_name, book_title):
8         self.request_id = request_id
9         self.requester_name = requester_name
10        self.book_title = book_title
11
12    def __str__(self):
13        return f"[{self.request_id}, Requester: {self.requester_name}, Book: {self.book_title}]"
14
15 class NormalQueue:
16     """Simple FIFO queue for book requests"""
17     def __init__(self):
18         self.queue = deque()
19
20     def add_request(self, request):
21         """Add a request to the queue"""
22         self.queue.append(request)
23         print(f"Request added: {request}")
24
25     def remove_request(self):
26         """Remove and return the first request"""
27         if not self.queue:
28             print("Queue is empty!")
29             return None
30         request = self.queue.popleft()
31         print(f"Request processed: {request}")
32         return request
33
34     def display_queue(self):
35         """Display all requests in queue"""
36         if not self.queue:
37             print("Queue is empty!")
38             return
39         print(f"--- Normal Queue ---")
40         for i, req in enumerate(self.queue, 1):
41             print(f"{i}. {req}")
42
43     def size(self):
44         return len(self.queue)
45
46 # ===== PRIORITY QUEUE IMPLEMENTATION =====
47 class PriorityBookRequest:
48     """Book request with priority (Faculty > Student)"""
49     def __init__(self, request_id, requester_name, book_title, user_type):
50         self.request_id = request_id
51         self.requester_name = requester_name
52         self.book_title = book_title
53         self.user_type = user_type # "Faculty" or "Student"
54
55     def __str__(self):
56         return f"[{self.request_id}, Requester: {self.requester_name} ({self.user_type}), Book: {self.book_title}]"
57
58     def get_priority(self):
59         """Return priority (lower number = higher priority)"""
60         if self.user_type.lower() == "faculty":
61             return 1
62         else:
63             return 2
```

```
1 class PriorityQueue:
2     """Priority Queue for book requests (Priority Queue)"""
3     def __init__(self):
4         self.queue = []
5
6     def add_request(self, request):
7         """Add a request with priority sorting"""
8         self.queue.append(request)
9         self.queue.sort(key=lambda x: x.get('priority'))
10        print(f"Request added: {request}")
11
12    def remove_request(self):
13        """Remove and return the highest priority request"""
14        if not self.queue:
15            print("Queue is empty")
16            return None
17        request = self.queue.pop(0)
18        print(f"Request processed: {request}")
19        return request
20
21    def display_queue(self):
22        """Display all requests in priority order"""
23        if not self.queue:
24            print("Queue is empty")
25            return
26        print("===== Priority Queue =====")
27        for i, req in enumerate(self.queue, 1):
28            print(f"{i}. {req}")
29
30    def size(self):
31        return len(self.queue)
32
33    # ===== DEMO PROGRAM =====
34    if __name__ == "__main__":
35        print("===== LIBRARY BOOK REQUEST SYSTEM =====")
36
37        # Normal Queue Demo
38        print("\n--- NORMAL QUEUE (FIFO) ---")
39        normal_q = PriorityQueue()
40
41        # Add requests
42        normal_q.add_request(bookRequest(1, "Alice", "Python Programming"))
43        normal_q.add_request(bookRequest(2, "Bob", "Data Science"))
44        normal_q.add_request(bookRequest(3, "Charlie", "Web Development"))
45        normal_q.add_request(bookRequest(4, "Diana", "Machine Learning"))
46
47        normal_q.display_queue()
48        print(f"Queue size: {normal_q.size()}")
49
50        # Process requests
51        print("\n--- Processing Requests (Normal Queue) ---")
52        normal_q.remove_request()
53        normal_q.remove_request()
54
55        normal_q.display_queue()
56        print(f"Queue size: {normal_q.size()}")
57
58        # Priority Queue Demo
59        print("\n--- PRIORITY QUEUE (Faculty First) ---")
60        priority_q = PriorityQueue()
61
62        # Add requests (mix of faculty and student)
63        priority_q.add_request(PriorityBookRequest(1, "Alice", "Python Programming", "Student"))
64        priority_q.add_request(PriorityBookRequest(2, "Prof. Bob", "Data Science", "Faculty"))
65        priority_q.add_request(PriorityBookRequest(3, "Charlie", "Web Development", "Student"))
66        priority_q.add_request(PriorityBookRequest(4, "Prof. Diana", "Machine Learning", "Faculty"))
67        priority_q.add_request(PriorityBookRequest(5, "Eve", "Databases", "Student"))
68
69        priority_q.display_queue()
70        print(f"Queue size: {priority_q.size()}")
71
72        # Process requests
73        print("\n--- Processing Requests (Priority Queue) ---")
74        priority_q.remove_request()
75        priority_q.remove_request()
76        priority_q.remove_request()
77
78        priority_q.display_queue()
79        print(f"Queue size: {priority_q.size()}")
```

```
1 class PriorityQueue:
2     """Priority Queue for book requests (Priority Queue)"""
3     def __init__(self):
4         self.queue = []
5
6     def add_request(self, request):
7         """Add a request with priority sorting"""
8         self.queue.append(request)
9         self.queue.sort(key=lambda x: x.get('priority'))
10        print(f"Request added: {request}")
11
12    def remove_request(self):
13        """Remove and return the highest priority request"""
14        if not self.queue:
15            print("Queue is empty")
16            return None
17        request = self.queue.pop(0)
18        print(f"Request processed: {request}")
19        return request
20
21    def display_queue(self):
22        """Display all requests in priority order"""
23        if not self.queue:
24            print("Queue is empty")
25            return
26        print("===== Priority Queue =====")
27        for i, req in enumerate(self.queue, 1):
28            print(f"{i}. {req}")
29
30    def size(self):
31        return len(self.queue)
32
33    # ===== DEMO PROGRAM =====
34    if __name__ == "__main__":
35        print("===== LIBRARY BOOK REQUEST SYSTEM =====")
36
37        # Normal Queue Demo
38        print("\n--- NORMAL QUEUE (FIFO) ---")
39        normal_q = PriorityQueue()
40
41        # Add requests
42        normal_q.add_request(bookRequest(1, "Alice", "Python Programming"))
43        normal_q.add_request(bookRequest(2, "Bob", "Data Science"))
44        normal_q.add_request(bookRequest(3, "Charlie", "Web Development"))
45        normal_q.add_request(bookRequest(4, "Diana", "Machine Learning"))
46
47        normal_q.display_queue()
48        print(f"Queue size: {normal_q.size()}")
49
50        # Process requests
51        print("\n--- Processing Requests (Normal Queue) ---")
52        normal_q.remove_request()
53        normal_q.remove_request()
54
55        normal_q.display_queue()
56        print(f"Queue size: {normal_q.size()}")
57
58        # Priority Queue Demo
59        print("\n--- PRIORITY QUEUE (Faculty First) ---")
60        priority_q = PriorityQueue()
61
62        # Add requests (mix of faculty and student)
63        priority_q.add_request(PriorityBookRequest(1, "Alice", "Python Programming", "Student"))
64        priority_q.add_request(PriorityBookRequest(2, "Prof. Bob", "Data Science", "Faculty"))
65        priority_q.add_request(PriorityBookRequest(3, "Charlie", "Web Development", "Student"))
66        priority_q.add_request(PriorityBookRequest(4, "Prof. Diana", "Machine Learning", "Faculty"))
67        priority_q.add_request(PriorityBookRequest(5, "Eve", "Databases", "Student"))
68
69        priority_q.display_queue()
70        print(f"Queue size: {priority_q.size()}")
71
72        # Process requests
73        print("\n--- Processing Requests (Priority Queue) ---")
74        priority_q.remove_request()
75        priority_q.remove_request()
76        priority_q.remove_request()
77
78        priority_q.display_queue()
79        print(f"Queue size: {priority_q.size()}")
```

Output:

```
PS C:\VAI Assistant Coding> & C:/Users/edula/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/AI Assistant Coding/bug.py"

=== Priority Queue ===
1. ID: 3, Requester: Charlie (Student), Book: Web Development
2. ID: 5, Requester: Eve (Student), Book: Databases
Queue size: 2

=====
PS C:\VAI Assistant Coding> 
```

## Explanation:

- Queue (FIFO) → First request comes, first served. (If a student requests first, they get the book first.)
- Priority Queue → Faculty requests are served before students, even if they come later.
- enqueue() → Adds a request to the system.
- dequeue() → Removes and processes the next request.

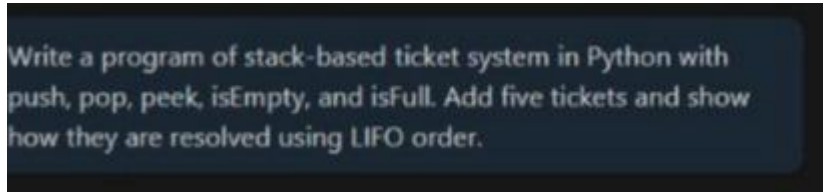
## Task 3: Emergency Help Desk (Stack Implementation)

### Scenario

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.

## Prompt:

Write a program of stack-based ticket system in Python with push, pop, peek, isEmpty, and isFull. Add five tickets and show how they are resolved using LIFO order.



## Code:

```
1 # ===== STACK-BASED TICKET SYSTEM =====
2
3 class Ticket:
4     """Represents a support ticket"""
5     def __init__(self, ticket_id, customer_name, issue):
6         self.ticket_id = ticket_id
7         self.customer_name = customer_name
8         self.issue = issue
9
10    def __str__(self):
11        return f"Ticket #{self.ticket_id} | Customer: {self.customer_name} | Issue: {self.issue}"
12
13 class TicketStack:
14     """Stack-based ticket management system (LIFO - Last In, First Out)"""
15     def __init__(self, max_size=10):
16         self.stack = []
17         self.max_size = max_size
18
19     def push(self, ticket):
20         """Add a ticket to the stack (top of stack)"""
21         if self.is_full():
22             print(f"Error: Stack is full! Cannot add ticket #{ticket.ticket_id}")
23             return False
24         self.stack.append(ticket)
25         print(f"/ Ticket added: {ticket}")
26         return True
27
28     def pop(self):
29         """Remove and return the ticket from the top of the stack"""
30         if self.is_empty():
31             print(f"Error: Stack is empty! No tickets to resolve.")
32             return None
33         ticket = self.stack.pop()
34         print(f"/ Resolving: {ticket}")
35         return ticket
36
37     def peek(self):
38         """View the top ticket without removing it"""
39         if self.is_empty():
40             print(f"Error: Stack is empty!")
41             return None
42         return self.stack[-1]
43
44     def is_empty(self):
45         """Check if the stack is empty"""
46         return len(self.stack) == 0
47
48     def is_full(self):
49         """Check if the stack is full"""
50         return len(self.stack) >= self.max_size
51
52     def size(self):
53         """Return the number of tickets in the stack"""
54         return len(self.stack)
55
56     def display_stack(self):
57         """Display all tickets in the stack (top to bottom)"""
58
59 # ===== Main Program =====
60
61 # Create a TicketStack instance
62 stack = TicketStack()
63
64 # Add five tickets
65 ticket1 = Ticket(1, "John Doe", "Login issue")
66 ticket2 = Ticket(2, "Jane Smith", "Password reset")
67 ticket3 = Ticket(3, "Mike Johnson", "Account locked")
68 ticket4 = Ticket(4, "Emily White", "Forgot email")
69 ticket5 = Ticket(5, "David Brown", "Two-factor auth")
70
71 stack.push(ticket1)
72 stack.push(ticket2)
73 stack.push(ticket3)
74 stack.push(ticket4)
75 stack.push(ticket5)
76
77 # Show the stack size
78 print(f"Stack size: {stack.size()}")
79
80 # Resolve tickets using LIFO order
81 while not stack.is_empty():
82     resolved_ticket = stack.pop()
83     print(f"Resolved ticket: {resolved_ticket}")
84
85 # Final stack size
86 print(f"Final stack size: {stack.size()}")
```



```
13 class TicketStack:
14     def __init__(self):
15         self.tickets = []
16         self.resolve_all_tickets()
17
18     def resolve_all_tickets(self):
19         """Resolve all tickets in LIFO order"""
20         print("\n" + "-" * 60)
21         print("RESOLVING ALL TICKETS (LIFO - Last In, First Out)")
22         print("-" * 60)
23         count = 1
24         while not self.is_empty():
25             self.resolve_next_ticket()
26             self.pop()
27             count += 1
28         print("\nAll tickets have been resolved!")
29         print("-" * 60)
30
31 # ===== DEMO PROGRAM =====
32 if __name__ == "__main__":
33     print("-" * 60)
34     print("STACK-BASED TICKET SYSTEM")
35     print("-" * 60)
36
37 # Create ticket stack with max size of 10
38 ticket_system = TicketStack(max_size=10)
39
40 # Add five tickets
41 print("\n--- ADDING TICKETS TO THE STACK ---\n")
42 ticket_system.push(ticket(101, "John Smith", "Login issue"))
43 ticket_system.push(ticket(102, "Sarah Johnson", "Payment processing error"))
44 ticket_system.push(ticket(103, "Mike Wilson", "Data export failed"))
45 ticket_system.push(ticket(104, "Emma Brown", "Account suspended"))
46 ticket_system.push(ticket(105, "David Lee", "Password reset not working"))
47
48 # Display current stack
49 ticket_system.display_stack()
50
51 # Show stack information
52 print("\nStack Size: (ticket_system.size())")
53 print("Is Stack Empty? (ticket_system.is_empty())")
54 print("Is Stack Full? (ticket_system.is_full())")
55
56 # Peek at the top ticket
57 print("\nPeek at top ticket: ----")
58 top_ticket = ticket_system.peek()
59 if top_ticket:
60     print(f"Top ticket (without removing): {top_ticket}")
61
62 # Resolve all tickets in LIFO order
63 ticket_system.resolve_all()
64
65 # Display final stack state
66 print("\nFinal Stack Size: (ticket_system.size())")
67 print("Is Stack Empty? (ticket_system.is_empty())")
68 ticket_system.display_stack()
69
70 # =====
```

Output:

```
PS C:\AI Assistant Coding\> cd "C:\AI Assistant Coding\bug.py"
(1) Resolving next ticket...
✓ Resolving Ticket #105 | Customer: David Lee | Issue: Password reset not working
✓ All tickets have been resolved!

Final Stack Size: 0
Is Stack Empty? True
Stack is empty! No tickets to display.
PS C:\AI Assistant Coding\>

(2) Resolving next ticket...
✓ Resolving Ticket #104 | Customer: Emma Brown | Issue: Account suspended
✓ Resolving Ticket #103 | Customer: Mike Wilson | Issue: Data export failed
✓ Resolving Ticket #102 | Customer: Sarah Johnson | Issue: Payment processing error
✓ Resolving Ticket #101 | Customer: John Smith | Issue: Login issue
✓ All tickets have been resolved!

(3) Resolving next ticket...
✓ Resolving Ticket #105 | Customer: David Lee | Issue: Password reset not working
✓ Resolving Ticket #104 | Customer: Emma Brown | Issue: Account suspended
✓ Resolving Ticket #103 | Customer: Mike Wilson | Issue: Data export failed
✓ Resolving Ticket #102 | Customer: Sarah Johnson | Issue: Payment processing error
✓ Resolving Ticket #101 | Customer: John Smith | Issue: Login issue
✓ All tickets have been resolved!

(4) Resolving next ticket...
✓ Resolving Ticket #105 | Customer: David Lee | Issue: Password reset not working
✓ Resolving Ticket #104 | Customer: Emma Brown | Issue: Account suspended
✓ Resolving Ticket #103 | Customer: Mike Wilson | Issue: Data export failed
✓ Resolving Ticket #102 | Customer: Sarah Johnson | Issue: Payment processing error
✓ Resolving Ticket #101 | Customer: John Smith | Issue: Login issue
✓ All tickets have been resolved!

(5) Resolving next ticket...
✓ Resolving Ticket #105 | Customer: David Lee | Issue: Password reset not working
✓ Resolving Ticket #104 | Customer: Emma Brown | Issue: Account suspended
✓ Resolving Ticket #103 | Customer: Mike Wilson | Issue: Data export failed
✓ Resolving Ticket #102 | Customer: Sarah Johnson | Issue: Payment processing error
✓ Resolving Ticket #101 | Customer: John Smith | Issue: Login issue
✓ All tickets have been resolved!

Final Stack Size: 0
Is Stack Empty? True
Stack is empty! No tickets to display.
PS C:\AI Assistant Coding\>
```

Explanation:

The program uses a stack to manage help desk tickets.

A stack works in last in, first solved order.

When a new ticket is raised, it is added to the top.  
When solving a ticket, the most recent one is handled first.

The program can also check if there are no tickets left or if the stack is full.

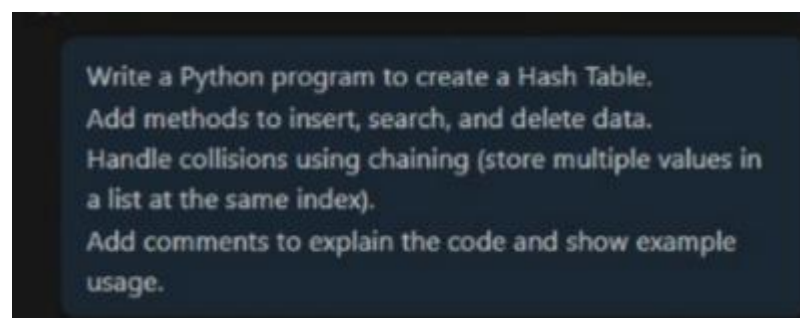
## **Task 4: Hash Table**

### **Objective**

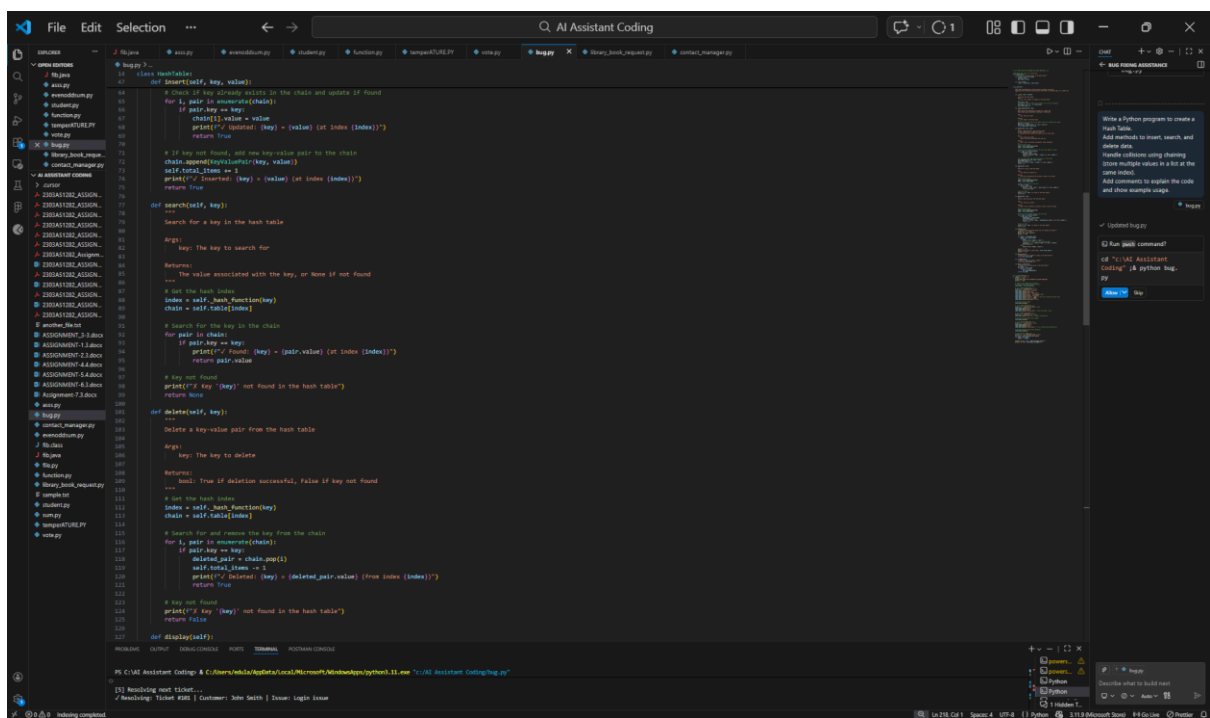
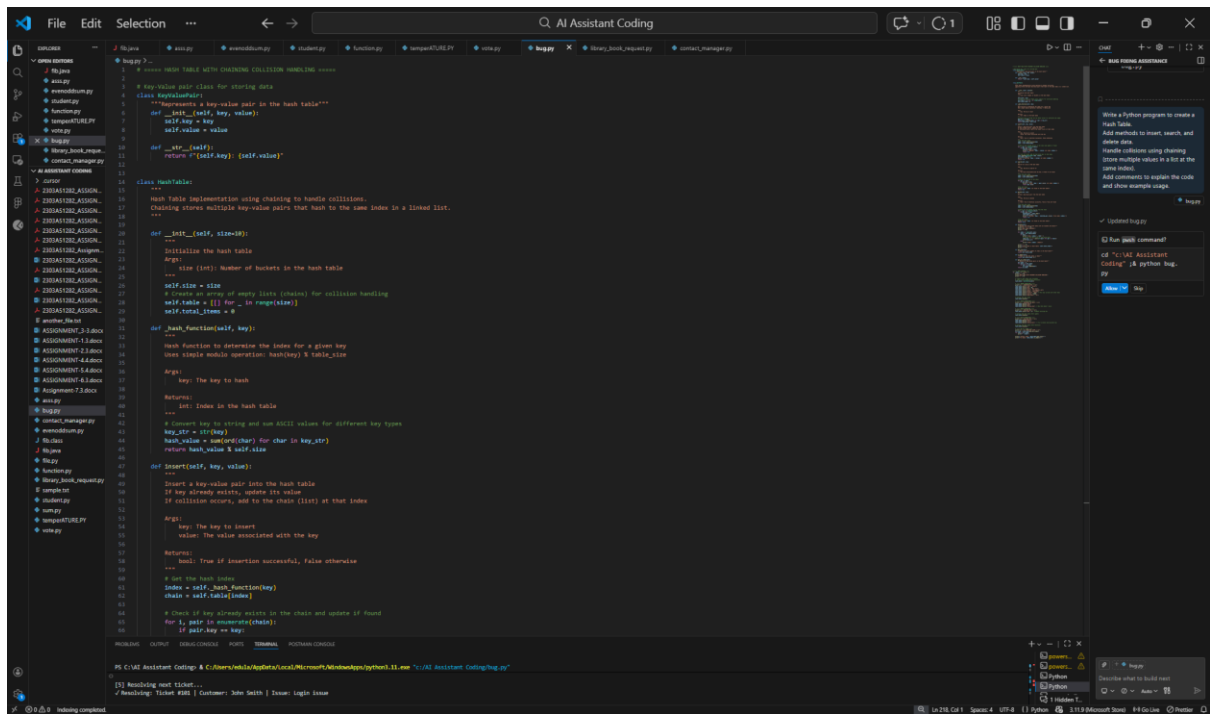
To implement a Hash Table and understand collision handling.

### **Prompt:**

Write a Python program to create a Hash Table.  
Add methods to insert, search, and delete data.  
Handle collisions using chaining (store multiple values in a list at the same index).  
Add comments to explain the code and show example usage.



### **Code:**



```
def get_all_items(self):  
    """Returns all key-value pairs in the hash table"""  
    all_items = []  
    for chain in self.table:  
        for pair in chain:  
            all_items.append(pair)  
    return all_items  
  
# ===== DEMO PROGRAM =====  
if __name__ == "__main__":  
    print("\n=====")  
    print("HASH TABLE WITH CHAINING COLLISION HANDLING")  
    print("=====")  
  
    # Create a hash table with 5 buckets  
    # (will also be demonstrated with collisions)  
    hash_table = HashTable(5)  
  
    # ===== INSERT OPERATIONS =====  
    print("\n--- INSERTING DATA ---")  
    hash_table.insert("name", "Alice")  
    hash_table.insert("age", 30)  
    hash_table.insert("city", "New York")  
    hash_table.insert("email", "alice@mail.com")  
    hash_table.insert("phone", "555-1234")  
    hash_table.insert("country", "USA") # This may collide with other keys  
    hash_table.insert("salary", 75000)  
    hash_table.insert("marital_status", "M")  
  
    # Display the hash table  
    hash_table.display()  
  
    # ===== SEARCH OPERATIONS =====  
    print("\n--- SEARCHING FOR DATA ---")  
    hash_table.search("name")  
    hash_table.search("age")  
    hash_table.search("unknown_key") # Key that doesn't exist  
  
    # ===== UPDATE OPERATIONS =====  
    print("\n--- UPDATING DATA ---")  
    hash_table.update("age", 35) # Update existing key  
  
    # Display the hash table after update  
    hash_table.display()  
  
    # ===== DELETE OPERATIONS =====  
    print("\n--- DELETING DATA ---")  
    hash_table.delete("email")  
    hash_table.delete("city")  
    hash_table.delete("nonexistent") # Try to delete non-existent key  
  
    # Display the hash table after deletions  
    hash_table.display()  
  
    # ===== GET ALL ITEMS =====  
    print("\n--- ALL EXISTING ITEMS ---")  
    all_items = hash_table.get_all_items()  
    for item in all_items:  
        print(f"Item: {item}")  
  
    print(f"Total items: {len(hash_table.get_all_items())}")  
    print(f"Is empty: {hash_table.is_empty()}")
```

## Output:

```
Index 0: age: 30 -> city: New York -> department: IT  
Index 1: name: Alice -> salary: 75000  
Index 2: phone: 555-1234 -> country: USA  
Index 3: [empty]  
Index 4: [empty]  
Total items in hash table: 5  
  
--- SEARCHING FOR DATA ---  
Found: name = Alice (at Index 2)  
Found: age = 30 (at Index 1)  
Key 'unknown_key' not found in the hash table  
  
--- UPDATING DATA ---  
Updated: age = 31 (at Index 1)  
  
HIGH TABLE CONTENTS  
Index 0: email: alice@mail.com  
Index 1: age: 31 -> city: New York -> department: IT  
Index 2: name: Alice -> salary: 75000  
Index 3: phone: 555-1234 -> country: USA  
Index 4: [empty]  
Total items in hash table: 5  
  
--- DELETING DATA ---  
Deleted: email = alice@mail.com (from Index 0)  
Deleted: city = New York (from Index 1)  
Key 'nonexistent' not found in the hash table  
  
HIGH TABLE CONTENTS  
Index 0: [empty]  
Index 1: age: 31 -> department: IT  
Index 2: name: Alice -> salary: 75000  
Index 3: phone: 555-1234 -> country: USA  
Index 4: [empty]  
Total items in hash table: 4  
  
--- ALL REMAINING ITEMS ---  
age: 31  
department: IT  
name: Alice  
salary: 75000  
phone: 555-1234  
country: USA  
Total items: 4  
Is empty: False  
PS C:\AI Assistant Coding>
```

## Explanation:

- A Hash Table stores data using a key and value.
- A hash function decides where to store the data.

- Sometimes two keys go to the same place. This is called a collision.
- To solve collisions, we use chaining, meaning we store multiple items in a list at the same index.
- The program should allow adding, finding, and removing data correctly.

### **Task 5: Real-Time Application Challenge**

#### Scenario

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

#### **Prompt:**

Create a Campus Resource Management System in Python. For each feature (Attendance, Event Registration, Library, Bus Schedule, Cafeteria Orders), choose the best data structure

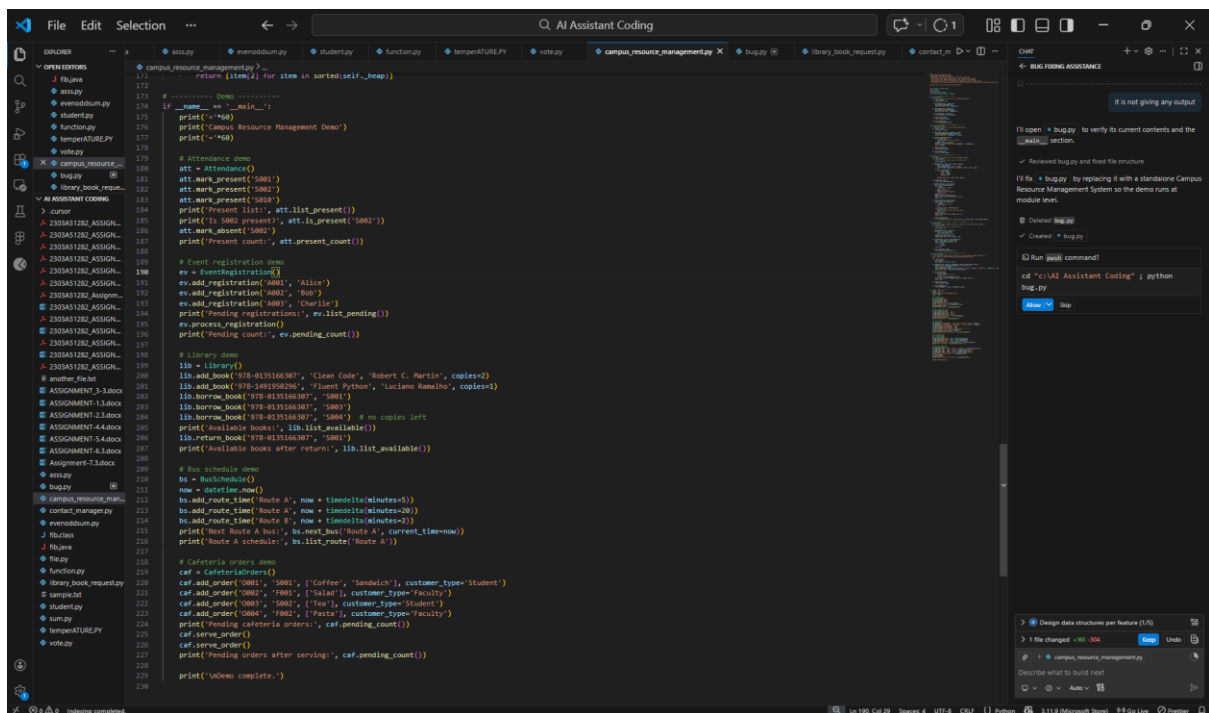
Create a Campus Resource Management System in Python. For each feature (Attendance, Event Registration, Library, Bus Schedule, Cafeteria Orders), choose the best data structure

## Code:

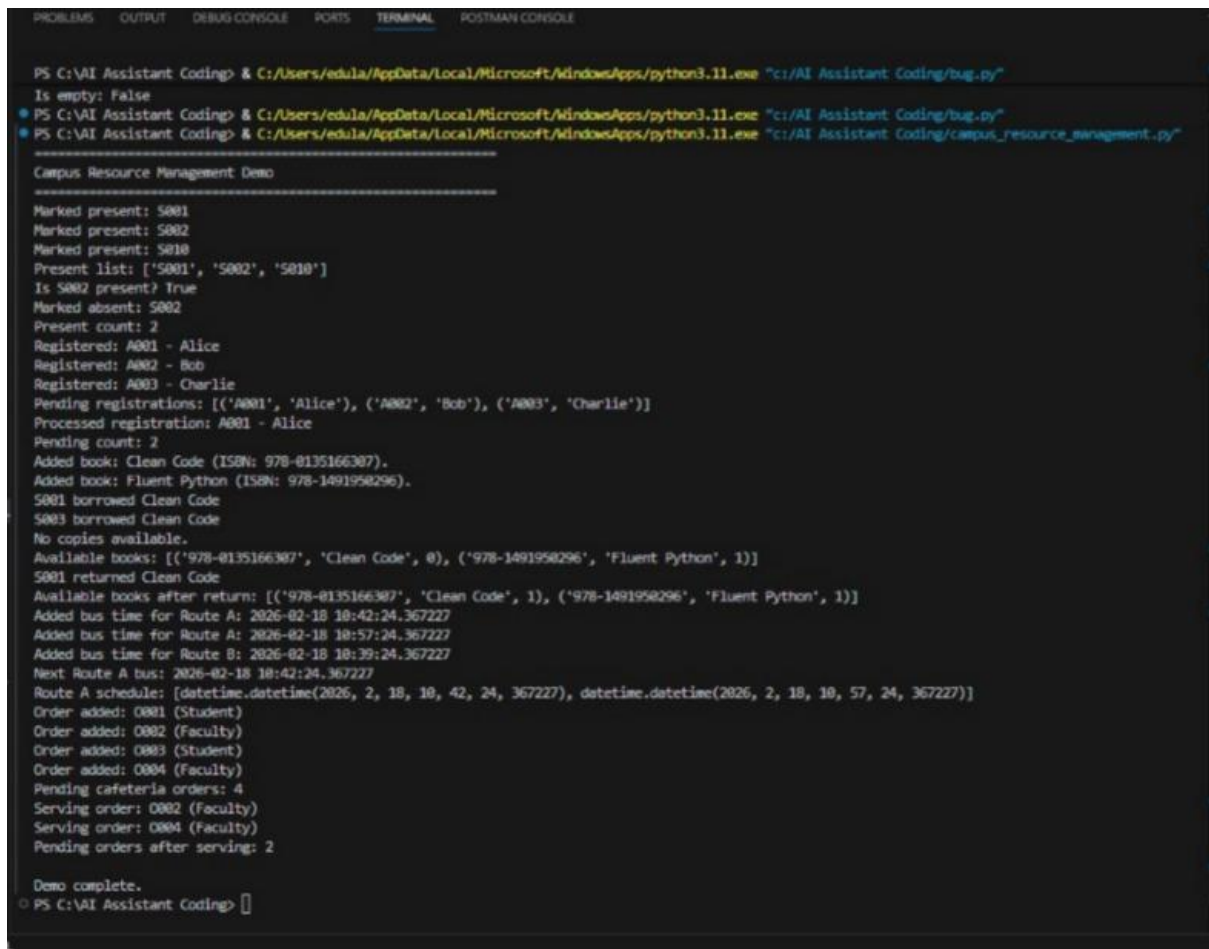
The image shows a VS Code editor window with a Python project. The left sidebar displays a file explorer with a project structure. The main editor shows the 'campus\_resource\_management.py' file, which contains Python code for managing resources. The code includes classes for 'Attendee', 'EventRegistration', and 'Student'. The right sidebar shows a 'Run and Debug' panel with a 'Run' button and a 'Debug' button. The bottom status bar shows 'Python 3.11.6 Microsoft Windows'.

```
File Edit Selection ... Q AI Assistant Coding
1 # flake8
2 Campus Resource Management System
3 Features and chosen data structures:
4 - Attendance: set (O(1) add/check) to track present student IDs
5 - Event Registration: queue (FIFO queue) for fair registration processing
6 - Library: dict (hash table) mapping ISBN -> book record for fast lookup
7 - Bus Schedule: dict of route -> sorted list of departure times (list kept sorted)
8 - Cafeteria Orders: heapq (priority queue) to prioritize faculty over students while preserving arrival order
9
10 Run this file to see a small demo of each feature.
11
12
13 from collections import deque
14 import heapq
15 import itertools
16 from bisect import insert
17 from datetime import datetime, timedelta
18
19 # ===== Attendance (set) =====
20 class Attendee:
21     """Track attendance using a set for O(1) add/remove/check."""
22     def __init__(self):
23         self.present = set()
24
25     def mark_present(self, student_id):
26         self.present.add(student_id)
27         print(f"Marked present: {student_id}")
28
29     def mark_absent(self, student_id):
30         self.present.discard(student_id)
31         print(f"Marked absent: {student_id}")
32
33     def is_present(self, student_id):
34         return student_id in self.present
35
36     def present_count(self):
37         return len(self.present)
38
39     def list_present(self):
40         return sorted(self.present)
41
42 # ===== Event Registration (FIFO queue) =====
43 class EventRegistration:
44     """Register attendees in arrival order using deque."""
45     def __init__(self):
46         self.queue = deque()
47
48     def add_registration(self, attendee_id, name):
49         self.queue.append((attendee_id, name))
50         print(f"Registered: {attendee_id} - {name}")
51
52     def process_registration(self):
53         if not self.queue:
54             return None
55         attendee = self.queue.popleft()
56         print(f"Processed registration: {attendee[0]} - {attendee[1]}")
57         return attendee
58
59     def pending_count(self):
60         return len(self.queue)
```

[illegible]



## Output:



## **Explanation:**

Library Book Borrowing using a queue:

- The queue stores student names who request a book.
- When a student requests a book, we use `enqueue()` to add them to the queue.
- When a book becomes available, we use `dequeue()` to give it to the first student in line.
- This ensures fairness because the first requester gets the book first.