Assignmennt 11.3 Ai Assisted Coding

Name: M.Sunil
Ht.no: 2303A51293

Batch: 05

Task 1:

Smart Contact Manager (Arrays & Linked Lists)

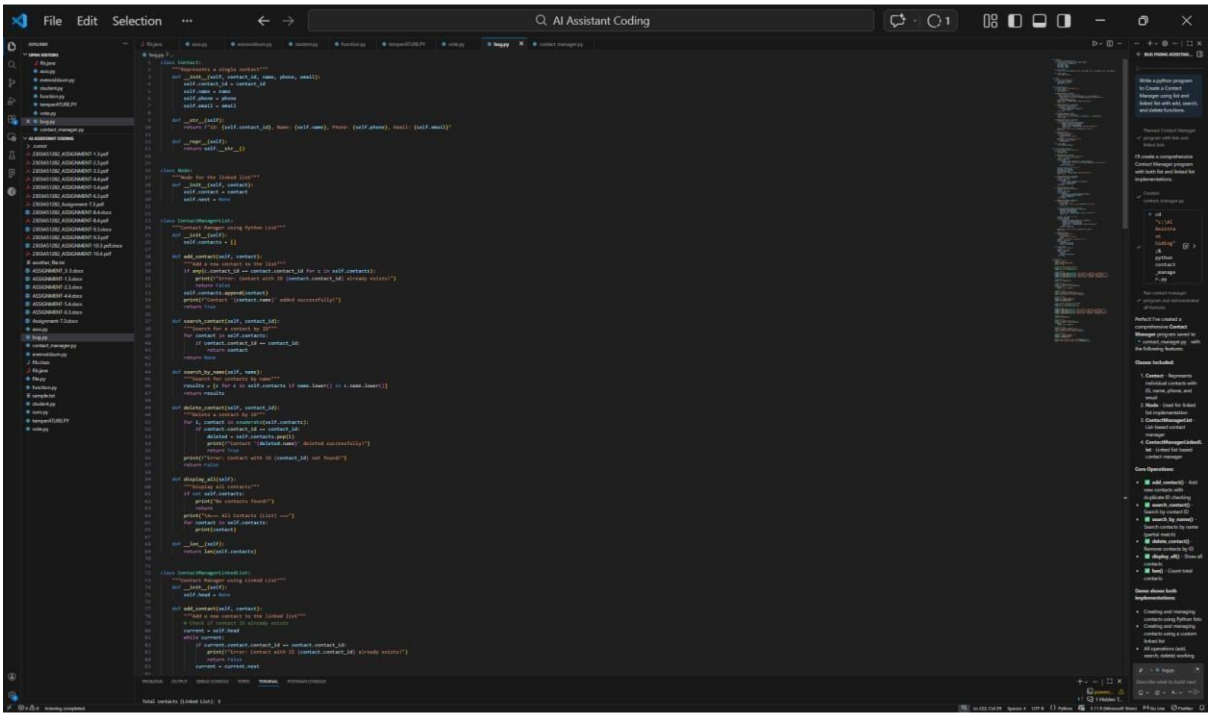
Scenario

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.

Code:



```python
class Contact:
    """Represents a single contact"""
    def __init__(self, contact_id, name, phone, email):
        self.contact_id = contact_id
        self.name = name
        self.phone = phone
        self.email = email

    def __str__(self):
        return f"ID: {self.contact_id}, Name: {self.name}, Phone: {self.phone}, Email: {self.email}"

    def __repr__(self):
        return self.__str__()


class Node:
    """Node for the linked list"""
    def __init__(self, contact):
        self.contact = contact
        self.next = None


class ContactManagerList:
    """Contact Manager using Python list"""
    def __init__(self):
        self.contacts = []

    def add_contact(self, contact):
        """Add a new contact to the list"""
        if any(c.contact_id == contact.contact_id for c in self.contacts):
            print(f"Error: Contact with ID {contact.contact_id} already exists!")
            return False
        self.contacts.append(contact)
        print(f"Contact '{contact.name}' added successfully!")
        return True

    def search_contact(self, contact_id):
        """Search for a contact by ID"""
        for contact in self.contacts:
            if contact.contact_id == contact_id:
                return contact
        return None

    def search_by_name(self, name):
        """Search for contacts by name"""
        results = [c for c in self.contacts if name.lower() in c.name.lower()]
        return results

    def delete_contact(self, contact_id):
        """Delete a contact by ID"""
        for i, contact in enumerate(self.contacts):
            if contact.contact_id == contact_id:
                deleted = self.contacts.pop(i)
                print(f"Contact '{deleted.name}' deleted successfully!")
                return True
        print(f"Error: Contact with ID {contact_id} not found!")
        return False

    def display_all(self):
        """Display all contacts"""
        if not self.contacts:
            print("No contacts found!")
            return
        print("\n--- All Contacts (List) ---")
        for contact in self.contacts:
            print(contact)

    def __len__(self):
        return len(self.contacts)


class ContactManagerLinkedList:
    """Contact Manager using Linked List"""
    def __init__(self):
        self.head = None

    def add_contact(self, contact):
        """Add a new contact to the linked list"""
        # Check if contact ID already exists
        current = self.head
        while current:
            if current.contact.contact_id == contact.contact_id:
                print(f"Error: Contact with ID {contact.contact_id} already exists!")
                return False
            current = current.next
```

```
Total contacts (Linked List): 3
```

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

Planned Contact Manager program with lists and linked lists

I'll create a comprehensive Contact Manager program with both list and linked list implementations.

Created contact_manager.py

```
cd "c:\AI Assistant Coding" ;& python contact_manager.py
```

Ran contact manager program and demonstrated all features

Perfect! I've created a comprehensive **Contact Manager** program saved to contact_manager.py with the following features:

Classes Included:

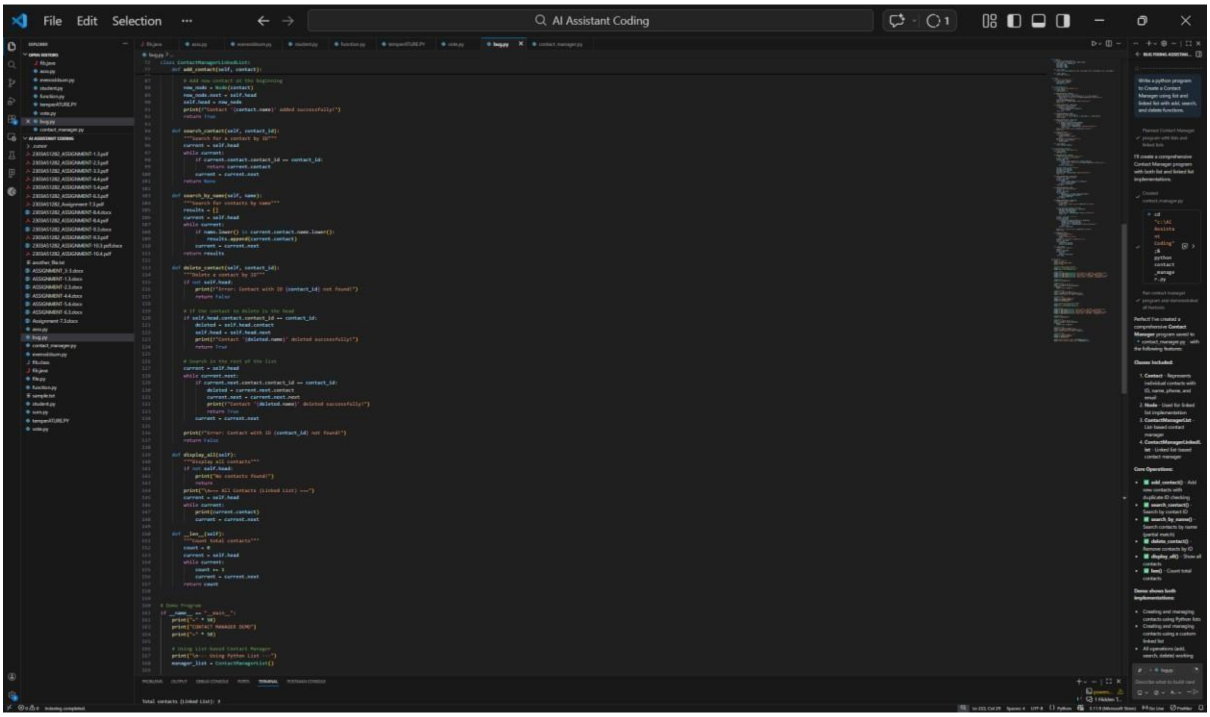
1. **Contact** - Represents individual contacts with ID, name, phone, and email
2. **Node** - Used for linked list implementation
3. **ContactManagerList** - List-based contact manager
4. **ContactManagerLinkedList** - Linked list-based contact manager

Core Operations:

- ✅ **add_contact()** - Add new contacts with duplicate ID checking
- ✅ **search_contact()** - Search by contact ID
- ✅ **search_by_name()** - Search contacts by name (partial match)
- ✅ **delete_contact()** - Remove contacts by ID
- ✅ **display_all()** - Show all contacts
- ✅ **len()** - Count total contacts

Demo shows both implementations:

- Creating and managing contacts using Python lists
- Creating and managing contacts using a custom linked list
- All operations (add, search, delete) working



```python
class ContactManagerLinkedList:
    def add_contact(self, contact):
        # Add new contact at the beginning
        new_node = Node(contact)
        new_node.next = self.head
        self.head = new_node
        print(f"Contact '{contact.name}' added successfully!")
        return True

    def search_contact(self, contact_id):
        """Search for a contact by ID"""
        current = self.head
        while current:
            if current.contact.contact_id == contact_id:
                return current.contact
            current = current.next
        return None

    def search_by_name(self, name):
        """Search for contacts by name"""
        results = []
        current = self.head
        while current:
            if name.lower() in current.contact.name.lower():
                results.append(current.contact)
            current = current.next
        return results

    def delete_contact(self, contact_id):
        """Delete a contact by ID"""
        if not self.head:
            print(f"Error: Contact with ID {contact_id} not found!")
            return False

        # If the contact to delete is the head
        if self.head.contact.contact_id == contact_id:
            deleted = self.head.contact
            self.head = self.head.next
            print(f"Contact '{deleted.name}' deleted successfully!")
            return True

        # Search in the rest of the list
        current = self.head
        while current.next:
            if current.next.contact.contact_id == contact_id:
                deleted = current.next.contact
                current.next = current.next.next
                print(f"Contact '{deleted.name}' deleted successfully!")
                return True
            current = current.next

        print(f"Error: Contact with ID {contact_id} not found!")
        return False

    def display_all(self):
        """Display all contacts"""
        if not self.head:
            print("No contacts found!")
            return
        print("\n=== All Contacts (Linked List) ===")
        current = self.head
        while current:
            print(current.contact)
            current = current.next

    def __len__(self):
        """Count total contacts"""
        count = 0
        current = self.head
        while current:
            count += 1
            current = current.next
        return count


# Demo Program
if __name__ == "__main__":
    print("=" * 50)
    print("CONTACT MANAGER DEMO")
    print("=" * 50)

    # Using List-based Contact Manager
    print("\n--- Using Python List ---")
    manager_list = ContactManagerList()
```

```
Total contacts (Linked List): 3
```

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

✓ Planned Contact Manager program with lists and linked lists

I'll create a comprehensive Contact Manager program with both list and linked list implementations.

✓ Created contact_manager.py

```
cd "c:\AI Assistant Coding" ;& python contact_manager.py
```

✓ Ran contact manager program and demonstrated all features

Perfect! I've created a comprehensive **Contact Manager** program saved to contact_manager.py with the following features:

Classes Included:

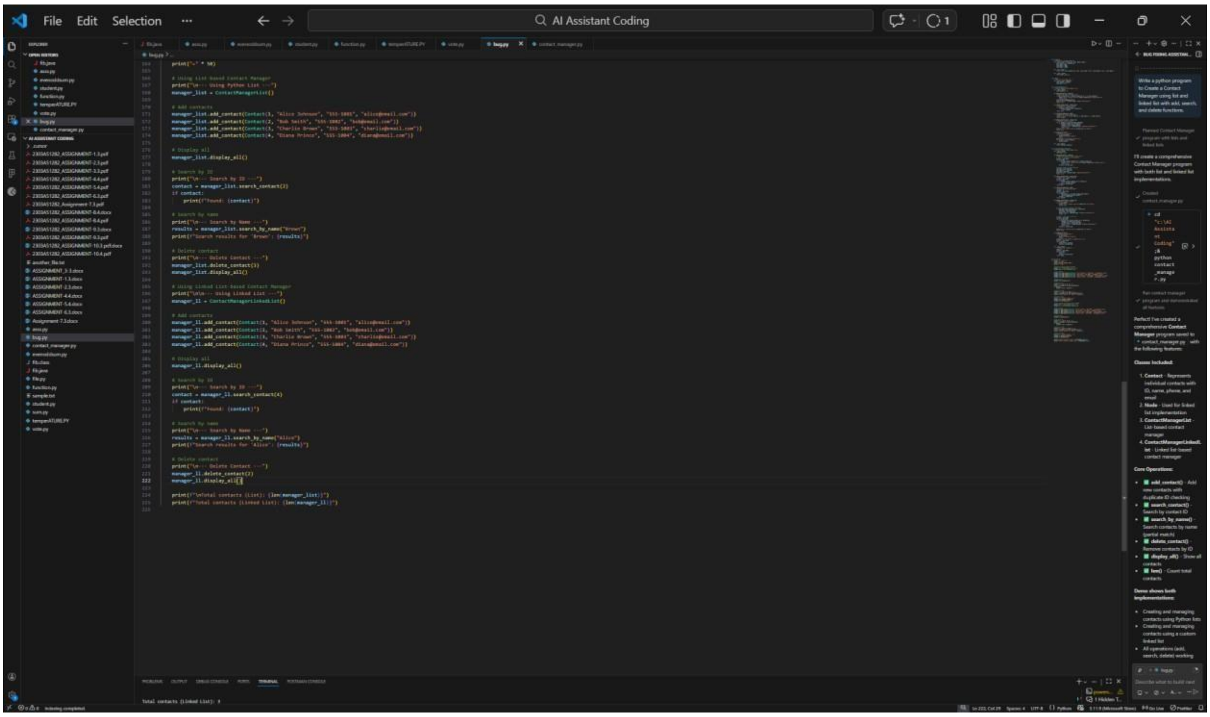
1. **Contact** - Represents individual contacts with ID, name, phone, and email
2. **Node** - Used for linked list implementation
3. **ContactManagerList** - List-based contact manager
4. **ContactManagerLinkedList** - Linked list-based contact manager

Core Operations:

- ✅ **add_contact()** - Add new contacts with duplicate ID checking
- ✅ **search_contact()** - Search by contact ID
- ✅ **search_by_name()** - Search contacts by name (partial match)
- ✅ **delete_contact()** - Remove contacts by ID
- ✅ **display_all()** - Show all contacts
- ✅ **len()** - Count total contacts

Demo shows both implementations:

- Creating and managing contacts using Python lists
- Creating and managing contacts using a custom linked list
- All operations (add, search, delete) working



```python
    print("=" * 50)

    # Using List-based Contact Manager
    print("\n--- Using Python List ---")
    manager_list = ContactManagerList()

    # Add contacts
    manager_list.add_contact(Contact(1, "Alice Johnson", "555-1001", "alice@email.com"))
    manager_list.add_contact(Contact(2, "Bob Smith", "555-1002", "bob@email.com"))
    manager_list.add_contact(Contact(3, "Charlie Brown", "555-1003", "charlie@email.com"))
    manager_list.add_contact(Contact(4, "Diana Prince", "555-1004", "diana@email.com"))

    # Display all
    manager_list.display_all()

    # Search by ID
    print("\n--- Search by ID ---")
    contact = manager_list.search_contact(2)
    if contact:
        print(f"Found: {contact}")

    # Search by name
    print("\n--- Search by Name ---")
    results = manager_list.search_by_name("Brown")
    print(f"Search results for 'Brown': {results}")

    # Delete contact
    print("\n--- Delete Contact ---")
    manager_list.delete_contact(3)
    manager_list.display_all()

    # Using Linked List-based Contact Manager
    print("\n\n--- Using Linked List ---")
    manager_ll = ContactManagerLinkedList()

    # Add contacts
    manager_ll.add_contact(Contact(1, "Alice Johnson", "555-1001", "alice@email.com"))
    manager_ll.add_contact(Contact(2, "Bob Smith", "555-1002", "bob@email.com"))
    manager_ll.add_contact(Contact(3, "Charlie Brown", "555-1003", "charlie@email.com"))
    manager_ll.add_contact(Contact(4, "Diana Prince", "555-1004", "diana@email.com"))

    # Display all
    manager_ll.display_all()

    # Search by ID
    print("\n--- Search by ID ---")
    contact = manager_ll.search_contact(4)
    if contact:
        print(f"Found: {contact}")

    # Search by name
    print("\n--- Search by Name ---")
    results = manager_ll.search_by_name("Alice")
    print(f"Search results for 'Alice': {results}")

    # Delete contact
    print("\n--- Delete Contact ---")
    manager_ll.delete_contact(2)
    manager_ll.display_all()

    print(f"\nTotal contacts (List): {len(manager_list)}")
    print(f"Total contacts (Linked List): {len(manager_ll)}")
```

```
Total contacts (Linked List): 3
```

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

✓ Planned Contact Manager program with lists and linked lists

I'll create a comprehensive Contact Manager program with both list and linked list implementations.

✓ Created contact_manager.py

cd "c:\AI Assistant Coding" ;& python contact_manager.py

✓ Ran contact manager program and demonstrated all features

Perfect! I've created a comprehensive Contact Manager program saved to contact_manager.py with the following features:

Classes Included:

1. Contact - Represents individual contacts with ID, name, phone, and email
2. Node - Used for linked list implementation
3. ContactManagerList - List-based contact manager
4. ContactManagerLinkedList - Linked list-based contact manager

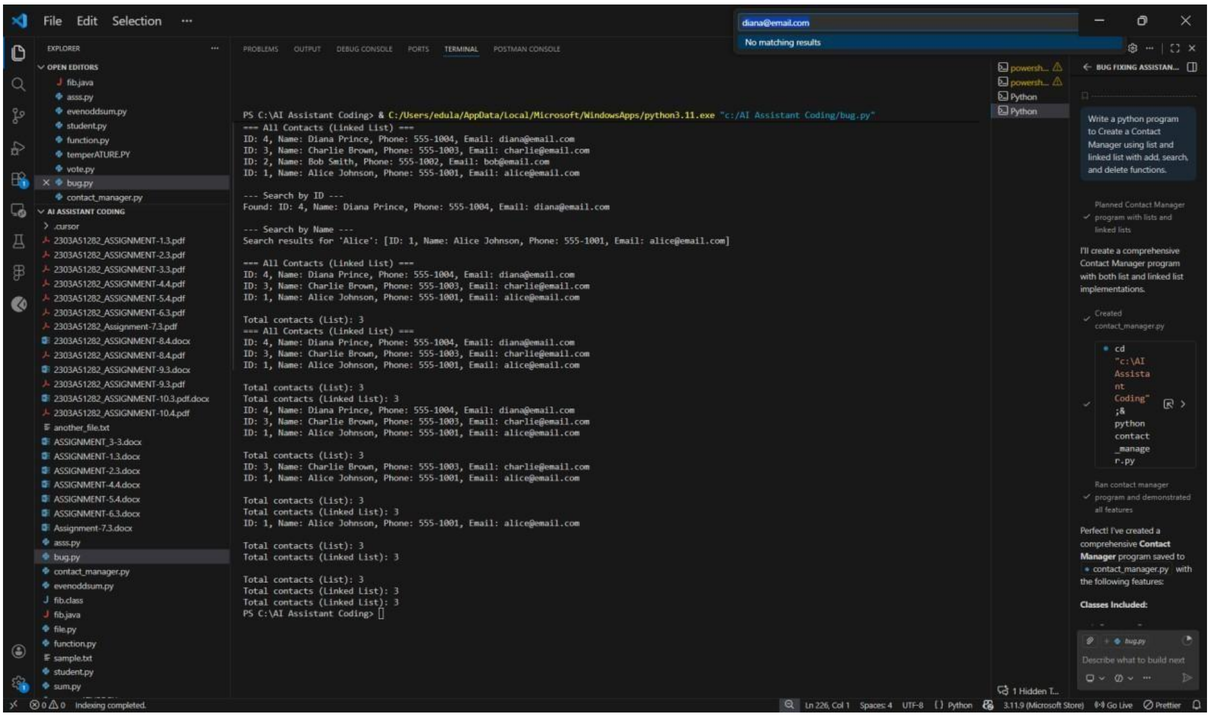
Core Operations:

- ✅ add_contact() - Add new contacts with duplicate ID checking
- ✅ search_contact() - Search by contact ID
- ✅ search_by_name() - Search contacts by name (partial match)
- ✅ delete_contact() - Remove contacts by ID
- ✅ display_all() - Show all contacts
- ✅ len() - Count total contacts

Demo shows both implementations:

- Creating and managing contacts using Python lists
- Creating and managing contacts using a custom linked list
- All operations (add, search, delete) working

Output:



```
PS C:\AI Assistant Coding> & C:/Users/edula/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/AI Assistant Coding/bug.py"
=== All Contacts (Linked List) ===
ID: 4, Name: Diana Prince, Phone: 555-1004, Email: diana@email.com
ID: 3, Name: Charlie Brown, Phone: 555-1003, Email: charlie@email.com
ID: 2, Name: Bob Smith, Phone: 555-1002, Email: bob@email.com
ID: 1, Name: Alice Johnson, Phone: 555-1001, Email: alice@email.com

--- Search by ID ---
Found: ID: 4, Name: Diana Prince, Phone: 555-1004, Email: diana@email.com

--- Search by Name ---
Search results for 'Alice': [ID: 1, Name: Alice Johnson, Phone: 555-1001, Email: alice@email.com]

=== All Contacts (Linked List) ===
ID: 4, Name: Diana Prince, Phone: 555-1004, Email: diana@email.com
ID: 3, Name: Charlie Brown, Phone: 555-1003, Email: charlie@email.com
ID: 1, Name: Alice Johnson, Phone: 555-1001, Email: alice@email.com

Total contacts (List): 3
=== All Contacts (Linked List) ===
ID: 4, Name: Diana Prince, Phone: 555-1004, Email: diana@email.com
ID: 3, Name: Charlie Brown, Phone: 555-1003, Email: charlie@email.com
ID: 1, Name: Alice Johnson, Phone: 555-1001, Email: alice@email.com

Total contacts (List): 3
Total contacts (Linked List): 3
ID: 4, Name: Diana Prince, Phone: 555-1004, Email: diana@email.com
ID: 3, Name: Charlie Brown, Phone: 555-1003, Email: charlie@email.com
ID: 1, Name: Alice Johnson, Phone: 555-1001, Email: alice@email.com

Total contacts (List): 3
ID: 3, Name: Charlie Brown, Phone: 555-1003, Email: charlie@email.com
ID: 1, Name: Alice Johnson, Phone: 555-1001, Email: alice@email.com

Total contacts (List): 3
Total contacts (Linked List): 3
ID: 1, Name: Alice Johnson, Phone: 555-1001, Email: alice@email.com

Total contacts (List): 3
Total contacts (Linked List): 3

Total contacts (List): 3
Total contacts (Linked List): 3
Total contacts (Linked List): 3
PS C:\AI Assistant Coding> 
```

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

✓ Planned Contact Manager program with lists and linked lists

I'll create a comprehensive Contact Manager program with both list and linked list implementations.

✓ Created contact_manager.py

```
cd "c:\AI Assistant Coding" ;& python contact_manager.py
```

✓ Ran contact manager program and demonstrated all features

Perfect! I've created a comprehensive **Contact Manager** program saved to contact_manager.py with the following features:

Classes Included:

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:

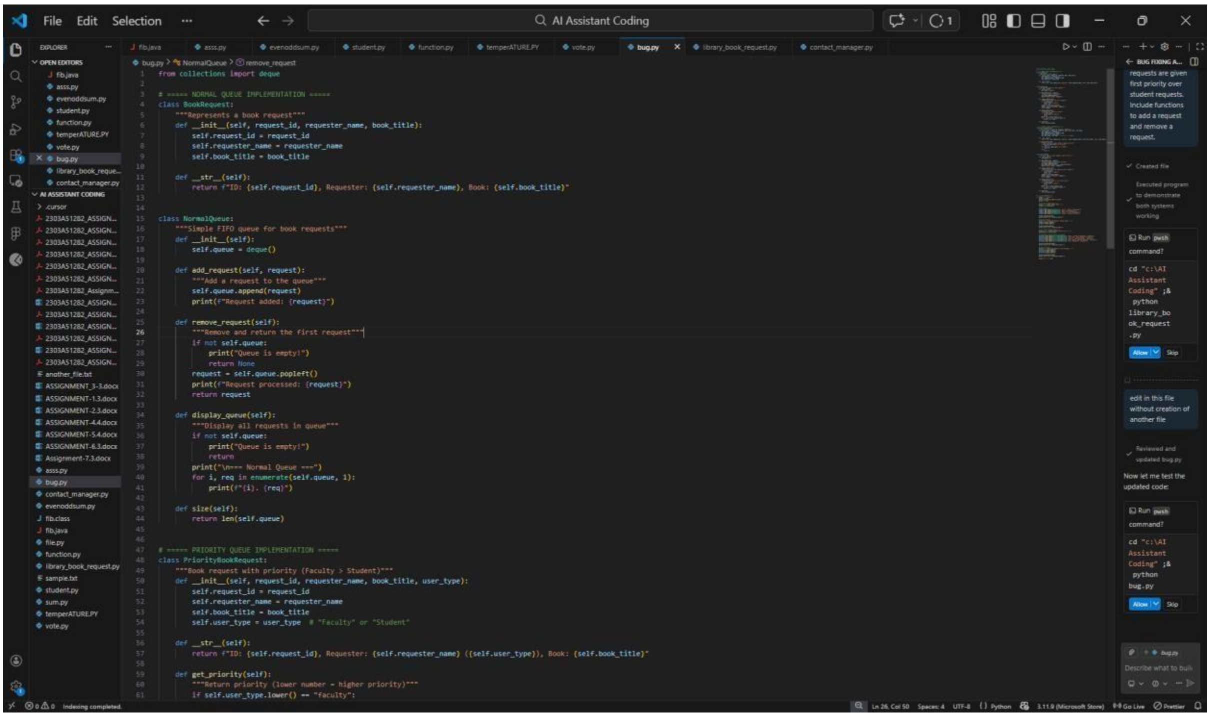
Library Book Search System (Queues & Priority Queues) Scenario

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.

Code:



```python
from collections import deque

# ===== NORMAL QUEUE IMPLEMENTATION =====
class BookRequest:
    """Represents a book request"""
    def __init__(self, request_id, requester_name, book_title):
        self.request_id = request_id
        self.requester_name = requester_name
        self.book_title = book_title

    def __str__(self):
        return f"ID: {self.request_id}, Requester: {self.requester_name}, Book: {self.book_title}"


class NormalQueue:
    """Simple FIFO queue for book requests"""
    def __init__(self):
        self.queue = deque()

    def add_request(self, request):
        """Add a request to the queue"""
        self.queue.append(request)
        print(f"Request added: {request}")

    def remove_request(self):
        """Remove and return the first request"""
        if not self.queue:
            print("Queue is empty!")
            return None
        request = self.queue.popleft()
        print(f"Request processed: {request}")
        return request

    def display_queue(self):
        """Display all requests in queue"""
        if not self.queue:
            print("Queue is empty!")
            return
        print("\n=== Normal Queue ===")
        for i, req in enumerate(self.queue, 1):
            print(f"{i}. {req}")

    def size(self):
        return len(self.queue)


# ===== PRIORITY QUEUE IMPLEMENTATION =====
class PriorityBookRequest:
    """Book request with priority (Faculty > Student)"""
    def __init__(self, request_id, requester_name, book_title, user_type):
        self.request_id = request_id
        self.requester_name = requester_name
        self.book_title = book_title
        self.user_type = user_type  # "Faculty" or "Student"

    def __str__(self):
        return f"ID: {self.request_id}, Requester: {self.requester_name} ({self.user_type}), Book: {self.book_title}"

    def get_priority(self):
        """Return priority (lower number = higher priority)"""
        if self.user_type.lower() == "faculty":
```

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

✓ Created file

✓ Executed program to demonstrate both systems working

Run pwsh command?

```
cd "c:\AI Assistant Coding" ;& python library_book_request.py
```

Allow | Skip

edit in this file without creation of another file

✓ Reviewed and updated bug.py

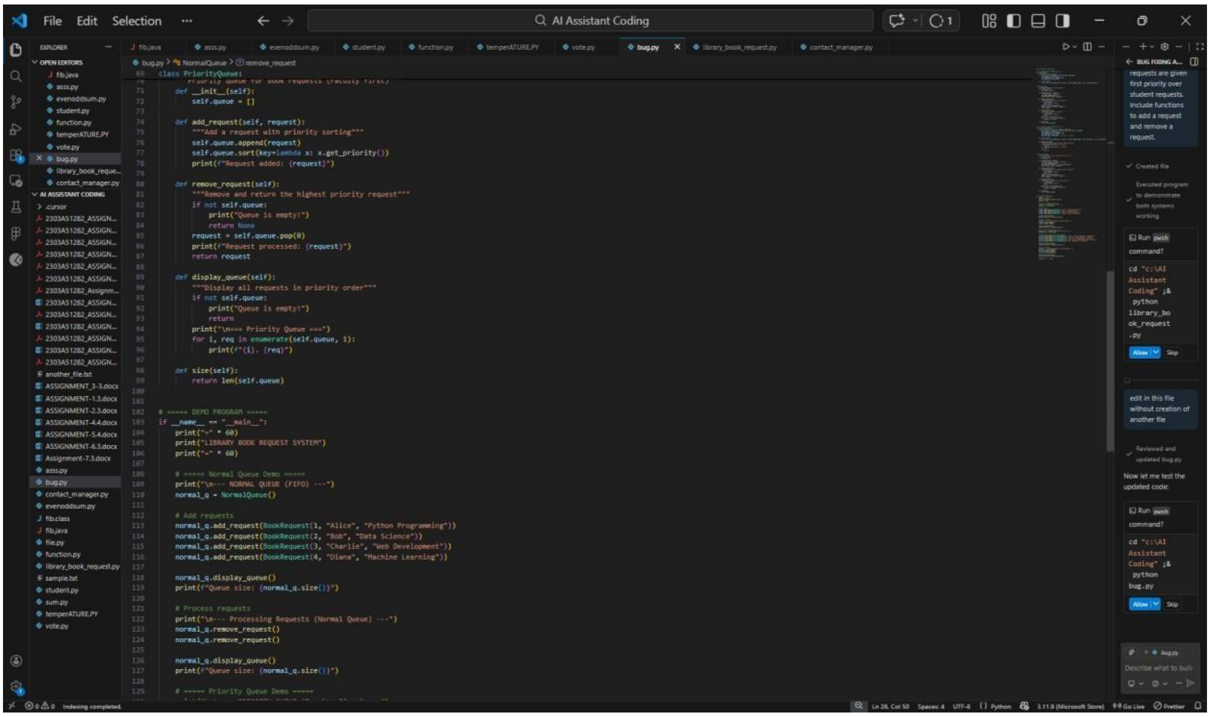
Now let me test the updated code:

Run pwsh command?

```
cd "c:\AI Assistant Coding" ;& python bug.py
```

Allow | Skip

Describe what to build



```python
class PriorityQueue:
    """Priority queue for book requests (Faculty first)"""
    def __init__(self):
        self.queue = []

    def add_request(self, request):
        """Add a request with priority sorting"""
        self.queue.append(request)
        self.queue.sort(key=lambda x: x.get_priority())
        print(f"Request added: {request}")

    def remove_request(self):
        """Remove and return the highest priority request"""
        if not self.queue:
            print("Queue is empty!")
            return None
        request = self.queue.pop(0)
        print(f"Request processed: {request}")
        return request

    def display_queue(self):
        """Display all requests in priority order"""
        if not self.queue:
            print("Queue is empty!")
            return
        print("\n=== Priority Queue ===")
        for i, req in enumerate(self.queue, 1):
            print(f"{i}. {req}")

    def size(self):
        return len(self.queue)


# ===== DEMO PROGRAM =====
if __name__ == "__main__":
    print("=" * 60)
    print("LIBRARY BOOK REQUEST SYSTEM")
    print("=" * 60)

    # ===== Normal Queue Demo =====
    print("\n--- NORMAL QUEUE (FIFO) ---")
    normal_q = NormalQueue()

    # Add requests
    normal_q.add_request(BookRequest(1, "Alice", "Python Programming"))
    normal_q.add_request(BookRequest(2, "Bob", "Data Science"))
    normal_q.add_request(BookRequest(3, "Charlie", "Web Development"))
    normal_q.add_request(BookRequest(4, "Diana", "Machine Learning"))

    normal_q.display_queue()
    print(f"Queue size: {normal_q.size()}")

    # Process requests
    print("\n--- Processing Requests (Normal Queue) ---")
    normal_q.remove_request()
    normal_q.remove_request()

    normal_q.display_queue()
    print(f"Queue size: {normal_q.size()}")

    # ===== Priority Queue Demo =====
```

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

✓ Created file

✓ Executed program to demonstrate both systems working

Run pwsh command?

cd "c:\AI Assistant Coding" ;& python library_book_request.py

Allow | Skip

edit in this file without creation of another file

✓ Reviewed and updated bug.py

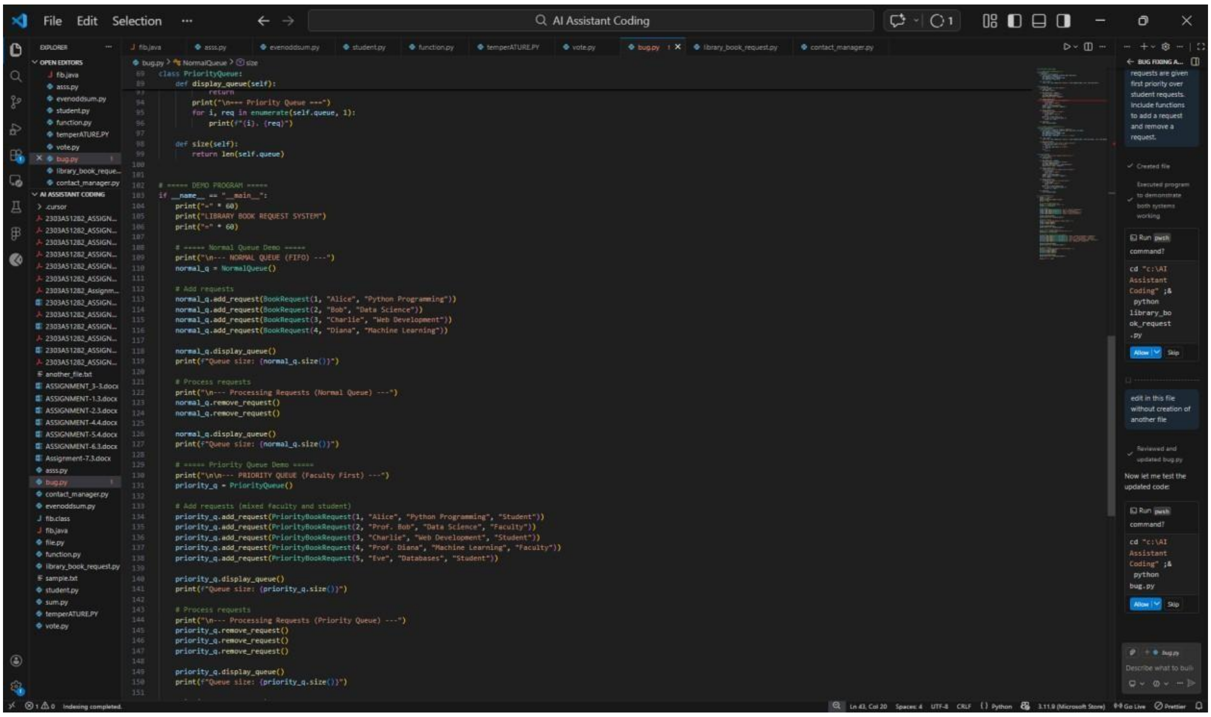
Now let me test the updated code:

Run pwsh command?

cd "c:\AI Assistant Coding" ;& python bug.py

Allow | Skip

Describe what to build



```
class PriorityQueue:
    def display_queue(self):
            return
        print("\n=== Priority Queue ===")
        for i, req in enumerate(self.queue, 1):
            print(f"{i}. {req}")

    def size(self):
        return len(self.queue)


# ===== DEMO PROGRAM =====
if __name__ == "__main__":
    print("=" * 60)
    print("LIBRARY BOOK REQUEST SYSTEM")
    print("=" * 60)

    # ===== Normal Queue Demo =====
    print("\n--- NORMAL QUEUE (FIFO) ---")
    normal_q = NormalQueue()

    # Add requests
    normal_q.add_request(BookRequest(1, "Alice", "Python Programming"))
    normal_q.add_request(BookRequest(2, "Bob", "Data Science"))
    normal_q.add_request(BookRequest(3, "Charlie", "Web Development"))
    normal_q.add_request(BookRequest(4, "Diana", "Machine Learning"))

    normal_q.display_queue()
    print(f"Queue size: {normal_q.size()}")

    # Process requests
    print("\n--- Processing Requests (Normal Queue) ---")
    normal_q.remove_request()
    normal_q.remove_request()

    normal_q.display_queue()
    print(f"Queue size: {normal_q.size()}")

    # ===== Priority Queue Demo =====
    print("\n\n--- PRIORITY QUEUE (Faculty First) ---")
    priority_q = PriorityQueue()

    # Add requests (mixed faculty and student)
    priority_q.add_request(PriorityBookRequest(1, "Alice", "Python Programming", "Student"))
    priority_q.add_request(PriorityBookRequest(2, "Prof. Bob", "Data Science", "Faculty"))
    priority_q.add_request(PriorityBookRequest(3, "Charlie", "Web Development", "Student"))
    priority_q.add_request(PriorityBookRequest(4, "Prof. Diana", "Machine Learning", "Faculty"))
    priority_q.add_request(PriorityBookRequest(5, "Eve", "Databases", "Student"))

    priority_q.display_queue()
    print(f"Queue size: {priority_q.size()}")

    # Process requests
    print("\n--- Processing Requests (Priority Queue) ---")
    priority_q.remove_request()
    priority_q.remove_request()
    priority_q.remove_request()

    priority_q.display_queue()
    print(f"Queue size: {priority_q.size()}")
```

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

✓ Created file

✓ Executed program to demonstrate both systems working

Run pwsh command?

```
cd "c:\AI Assistant Coding" ;& python library_book_request.py
```

Allow | Skip

edit in this file without creation of another file

✓ Reviewed and updated bug.py

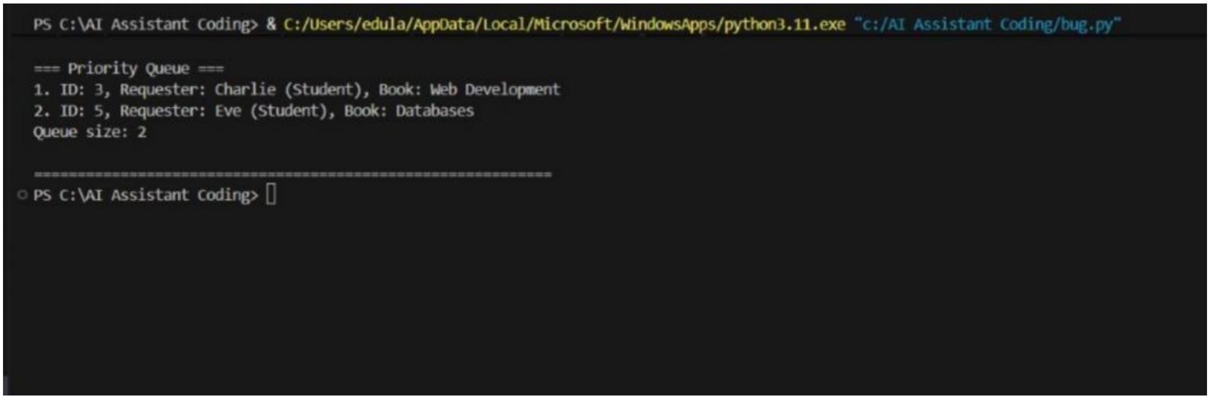
Now let me test the updated code:

Run pwsh command?

```
cd "c:\AI Assistant Coding" ;& python bug.py
```

Allow | Skip

Output:



```
PS C:\AI 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:\AI 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

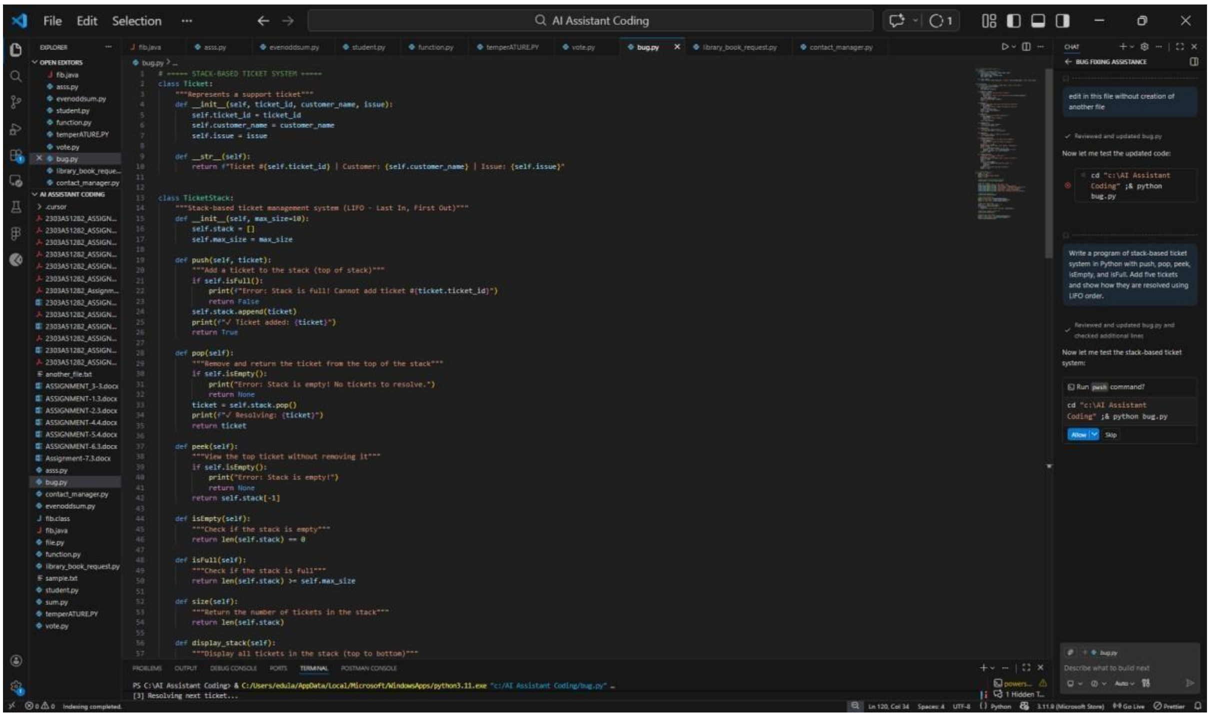
SR University's IT Help Desk receives technical support tickets from students and staff.

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:



```python
# ===== STACK-BASED TICKET SYSTEM =====
class Ticket:
    """Represents a support ticket"""
    def __init__(self, ticket_id, customer_name, issue):
        self.ticket_id = ticket_id
        self.customer_name = customer_name
        self.issue = issue

    def __str__(self):
        return f"Ticket #{self.ticket_id} | Customer: {self.customer_name} | Issue: {self.issue}"


class TicketStack:
    """Stack-based ticket management system (LIFO - Last In, First Out)"""
    def __init__(self, max_size=10):
        self.stack = []
        self.max_size = max_size

    def push(self, ticket):
        """Add a ticket to the stack (top of stack)"""
        if self.isFull():
            print(f"Error: Stack is full! Cannot add ticket #{ticket.ticket_id}")
            return False
        self.stack.append(ticket)
        print(f"✓ Ticket added: {ticket}")
        return True

    def pop(self):
        """Remove and return the ticket from the top of the stack"""
        if self.isEmpty():
            print("Error: Stack is empty! No tickets to resolve.")
            return None
        ticket = self.stack.pop()
        print(f"✓ Resolving: {ticket}")
        return ticket

    def peek(self):
        """View the top ticket without removing it"""
        if self.isEmpty():
            print("Error: Stack is empty!")
            return None
        return self.stack[-1]

    def isEmpty(self):
        """Check if the stack is empty"""
        return len(self.stack) == 0

    def isFull(self):
        """Check if the stack is full"""
        return len(self.stack) >= self.max_size

    def size(self):
        """Return the number of tickets in the stack"""
        return len(self.stack)

    def display_stack(self):
        """Display all tickets in the stack (top to bottom)"""
```

```
PS C:\AI Assistant Coding> & C:/Users/edula/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/AI Assistant Coding/bug.py"
[3] Resolving next ticket...
```

edit in this file without creation of another file

✓ Reviewed and updated bug.py

Now let me test the updated code:

```
cd "c:\AI Assistant Coding" ;& python bug.py
```

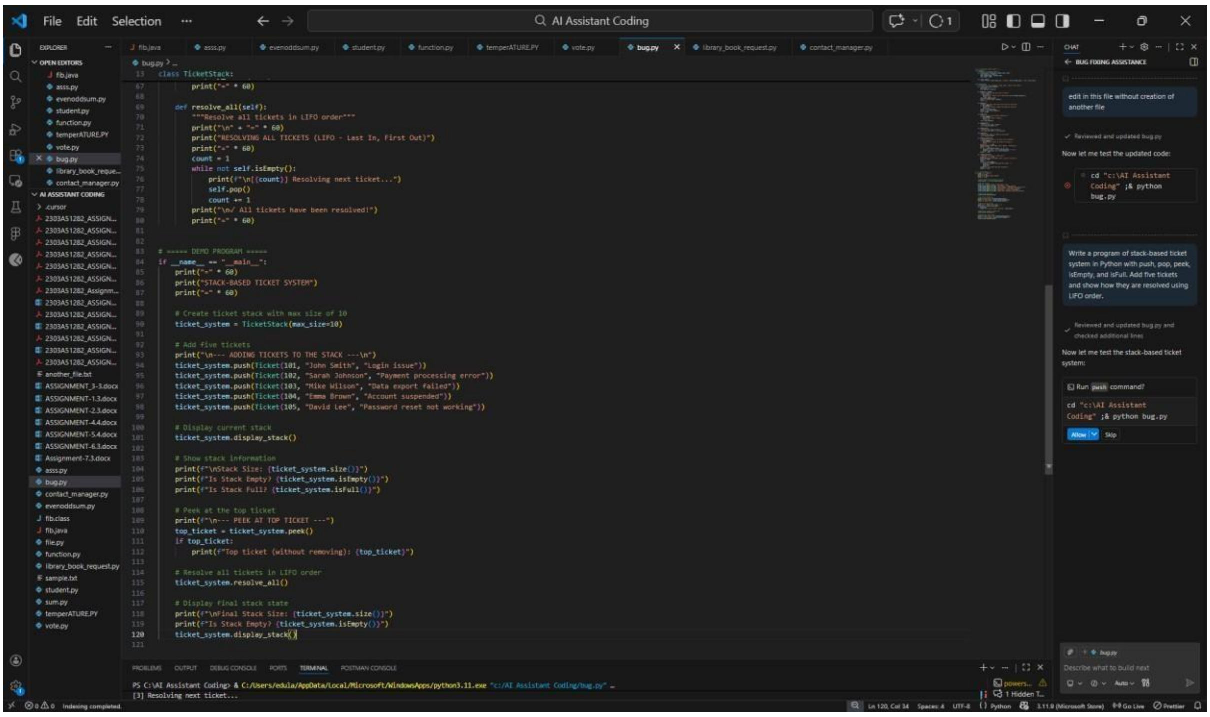
Write a program of stack-based ticket system in Python with push, pop, peek, isEmpty, and isFull. Add five tickets and show how they are resolved using LIFO order.

✓ Reviewed and updated bug.py and checked additional lines

Now let me test the stack-based ticket system:

Run pwsh command?

```
cd "c:\AI Assistant Coding" ;& python bug.py
```



```python
class TicketStack:
            print("=" * 60)

    def resolve_all(self):
        """Resolve all tickets in LIFO order"""
        print("\n" + "=" * 60)
        print("RESOLVING ALL TICKETS (LIFO - Last In, First Out)")
        print("=" * 60)
        count = 1
        while not self.isEmpty():
            print(f"\n[{count}] Resolving next ticket...")
            self.pop()
            count += 1
        print("\n✓ All tickets have been resolved!")
        print("=" * 60)


# ===== DEMO PROGRAM =====
if __name__ == "__main__":
    print("=" * 60)
    print("STACK-BASED TICKET SYSTEM")
    print("=" * 60)

    # Create ticket stack with max size of 10
    ticket_system = TicketStack(max_size=10)

    # Add five tickets
    print("\n--- ADDING TICKETS TO THE STACK ---\n")
    ticket_system.push(Ticket(101, "John Smith", "Login issue"))
    ticket_system.push(Ticket(102, "Sarah Johnson", "Payment processing error"))
    ticket_system.push(Ticket(103, "Mike Wilson", "Data export failed"))
    ticket_system.push(Ticket(104, "Emma Brown", "Account suspended"))
    ticket_system.push(Ticket(105, "David Lee", "Password reset not working"))

    # Display current stack
    ticket_system.display_stack()

    # Show stack information
    print(f"\nStack Size: {ticket_system.size()}")
    print(f"Is Stack Empty? {ticket_system.isEmpty()}")
    print(f"Is Stack Full? {ticket_system.isFull()}")

    # Peek at the top ticket
    print(f"\n--- PEEK AT TOP TICKET ---")
    top_ticket = ticket_system.peek()
    if top_ticket:
        print(f"Top ticket (without removing): {top_ticket}")

    # Resolve all tickets in LIFO order
    ticket_system.resolve_all()

    # Display final stack state
    print(f"\nFinal Stack Size: {ticket_system.size()}")
    print(f"Is Stack Empty? {ticket_system.isEmpty()}")
    ticket_system.display_stack()
```

```
PS C:\AI Assistant Coding> & C:/Users/edula/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/AI Assistant Coding/bug.py"
[3] Resolving next ticket...
```

edit in this file without creation of another file

✓ Reviewed and updated bug.py

Now let me test the updated code:

```
cd "c:\AI Assistant Coding" ;& python bug.py
```

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.

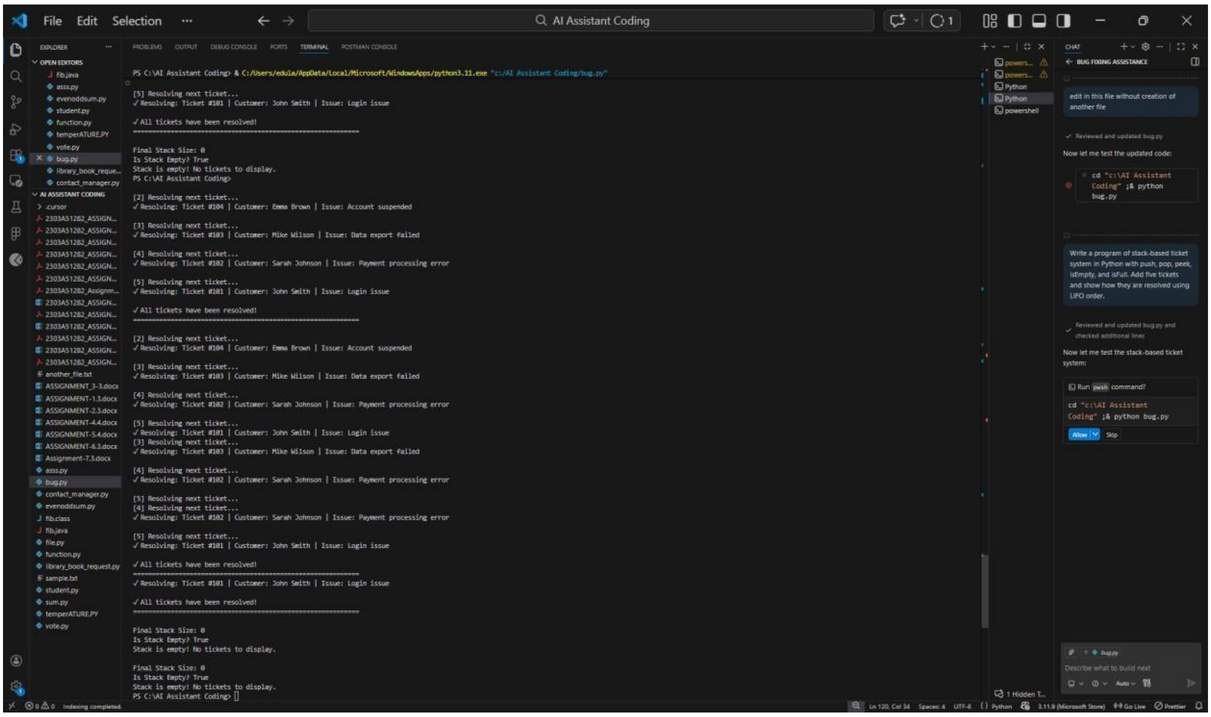
✓ Reviewed and updated bug.py and checked additional lines

Now let me test the stack-based ticket system:

Run pwsh command?

```
cd "c:\AI Assistant Coding" ;& python bug.py
```

Output:



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

[5] Resolving next ticket...
✓ 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>

[2] Resolving next ticket...
✓ Resolving: Ticket #104 | Customer: Emma Brown | Issue: Account suspended

[3] Resolving next ticket...
✓ Resolving: Ticket #103 | Customer: Mike Wilson | Issue: Data export failed

[4] Resolving next ticket...
✓ Resolving: Ticket #102 | Customer: Sarah Johnson | Issue: Payment processing error

[5] Resolving next ticket...
✓ Resolving: Ticket #101 | Customer: John Smith | Issue: Login issue

✓ All tickets have been resolved!
============================================================

[2] Resolving next ticket...
✓ Resolving: Ticket #104 | Customer: Emma Brown | Issue: Account suspended

[3] Resolving next ticket...
✓ Resolving: Ticket #103 | Customer: Mike Wilson | Issue: Data export failed

[4] Resolving next ticket...
✓ Resolving: Ticket #102 | Customer: Sarah Johnson | Issue: Payment processing error

[5] Resolving next ticket...
✓ Resolving: Ticket #101 | Customer: John Smith | Issue: Login issue
[3] Resolving next ticket...
✓ Resolving: Ticket #103 | Customer: Mike Wilson | Issue: Data export failed

[4] Resolving next ticket...
✓ Resolving: Ticket #102 | Customer: Sarah Johnson | Issue: Payment processing error

[5] Resolving next ticket...
[4] Resolving next ticket...
✓ Resolving: Ticket #102 | Customer: Sarah Johnson | Issue: Payment processing error

[5] Resolving next ticket...
✓ Resolving: Ticket #101 | Customer: John Smith | Issue: Login issue

✓ All tickets have been resolved!
============================================================
✓ 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.

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

BUG FIXING ASSISTANCE

edit in this file without creation of another file

✓ Reviewed and updated bug.py

Now let me test the updated code:

```
cd "c:\AI Assistant Coding" ;& python bug.py
```

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.

✓ Reviewed and updated bug.py and checked additional lines

Now let me test the stack-based ticket system:

Run push command?

```
cd "c:\AI Assistant Coding" ;& python bug.py
```

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:

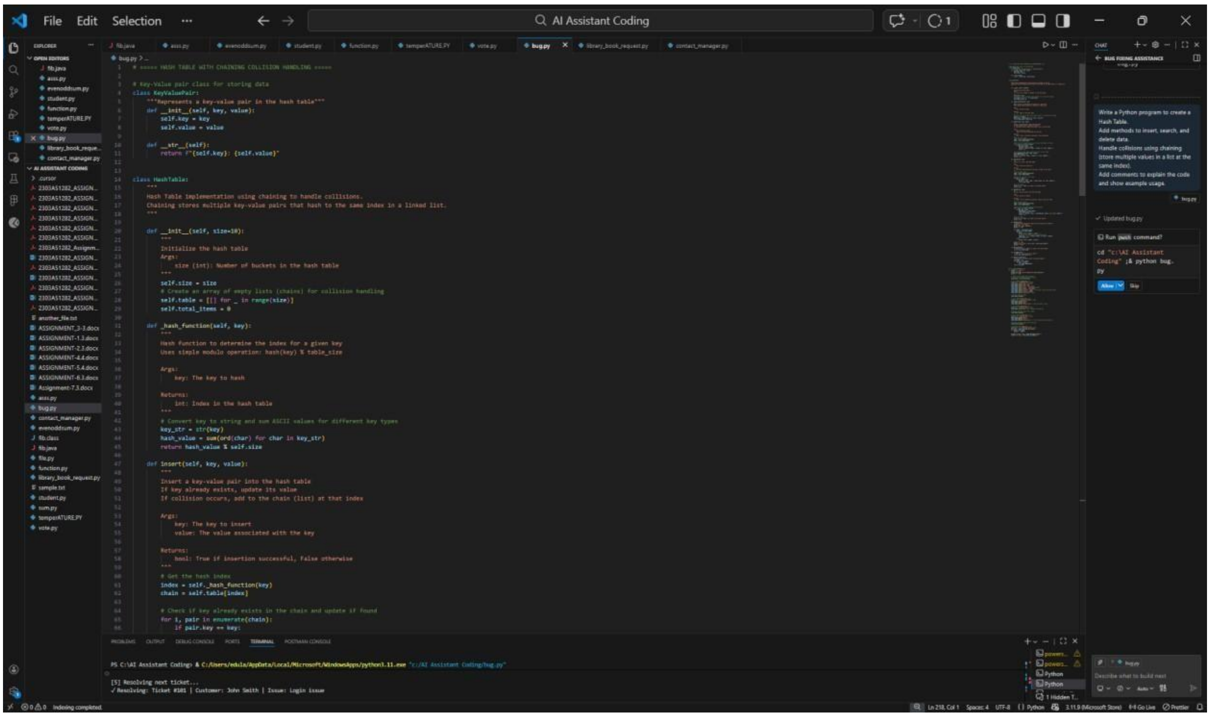
Write a Python program to create a Hash Table.

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:



```python
# ===== HASH TABLE WITH CHAINING COLLISION HANDLING =====

# Key-Value pair class for storing data
class KeyValuePair:
    """Represents a key-value pair in the hash table"""
    def __init__(self, key, value):
        self.key = key
        self.value = value

    def __str__(self):
        return f"{self.key}: {self.value}"


class HashTable:
    """
    Hash Table implementation using chaining to handle collisions.
    Chaining stores multiple key-value pairs that hash to the same index in a linked list.
    """

    def __init__(self, size=10):
        """
        Initialize the hash table
        Args:
            size (int): Number of buckets in the hash table
        """
        self.size = size
        # Create an array of empty lists (chains) for collision handling
        self.table = [[] for _ in range(size)]
        self.total_items = 0

    def _hash_function(self, key):
        """
        Hash function to determine the index for a given key
        Uses simple modulo operation: hash(key) % table_size

        Args:
            key: The key to hash

        Returns:
            int: Index in the hash table
        """
        # Convert key to string and sum ASCII values for different key types
        key_str = str(key)
        hash_value = sum(ord(char) for char in key_str)
        return hash_value % self.size

    def insert(self, key, value):
        """
        Insert a key-value pair into the hash table
        If key already exists, update its value
        If collision occurs, add to the chain (list) at that index

        Args:
            key: The key to insert
            value: The value associated with the key

        Returns:
            bool: True if insertion successful, False otherwise
        """
        # Get the hash index
        index = self._hash_function(key)
        chain = self.table[index]

        # Check if key already exists in the chain and update if found
        for i, pair in enumerate(chain):
            if pair.key == key:
```

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

[5] Resolving next ticket...
✓ Resolving: Ticket #101 | Customer: John Smith | Issue: Login issue
```

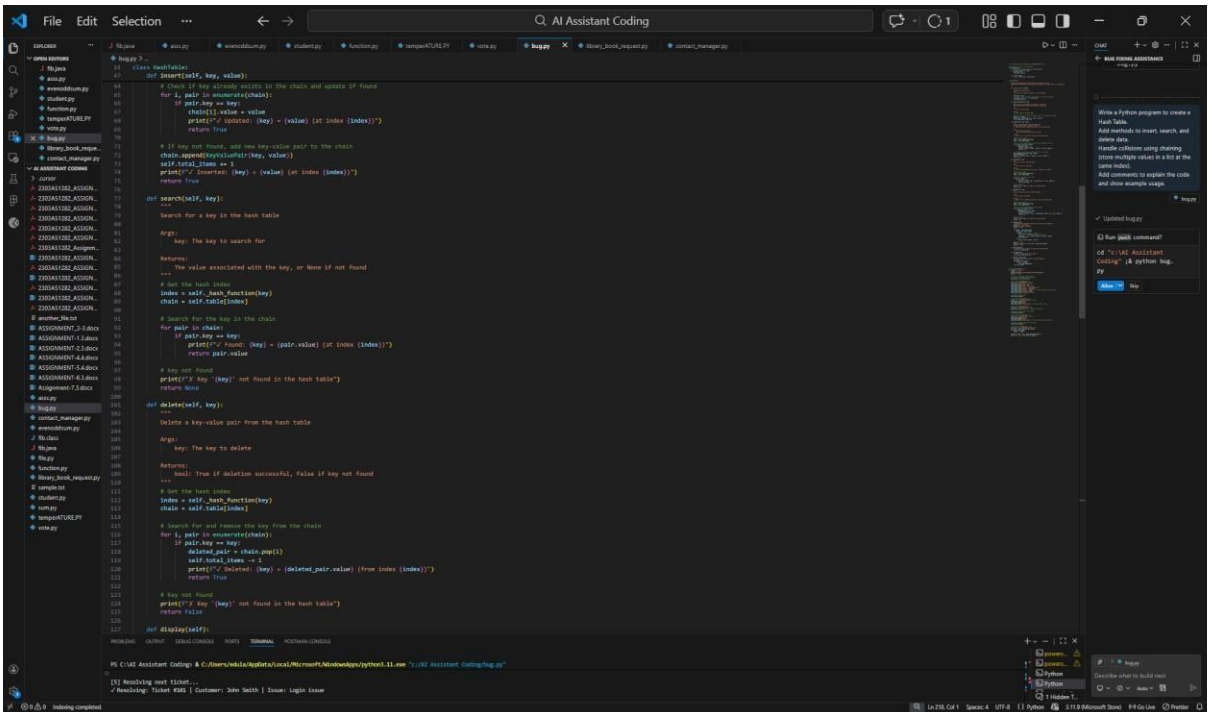
Write a Python program to create a Hash Table.
Add methods to insert, search, and delete data.
Handle collisions using chaining (store multiple values in a list at the same index).
Add comments to explain the code and show example usage.

✓ Updated bug.py

Run pwsh command?

```
cd "c:\AI Assistant Coding" ;& python bug.py
```

Allow | Skip



```python
class HashTable:
    def insert(self, key, value):
        # Check if key already exists in the chain and update if found
        for i, pair in enumerate(chain):
            if pair.key == key:
                chain[i].value = value
                print(f"✓ Updated: {key} = {value} (at index {index})")
                return True

        # If key not found, add new key-value pair to the chain
        chain.append(KeyValuePair(key, value))
        self.total_items += 1
        print(f"✓ Inserted: {key} = {value} (at index {index})")
        return True

    def search(self, key):
        """
        Search for a key in the hash table

        Args:
            key: The key to search for

        Returns:
            The value associated with the key, or None if not found
        """
        # Get the hash index
        index = self._hash_function(key)
        chain = self.table[index]

        # Search for the key in the chain
        for pair in chain:
            if pair.key == key:
                print(f"✓ Found: {key} = {pair.value} (at index {index})")
                return pair.value

        # Key not found
        print(f"X Key '{key}' not found in the hash table")
        return None

    def delete(self, key):
        """
        Delete a key-value pair from the hash table

        Args:
            key: The key to delete

        Returns:
            bool: True if deletion successful, False if key not found
        """
        # Get the hash index
        index = self._hash_function(key)
        chain = self.table[index]

        # Search for and remove the key from the chain
        for i, pair in enumerate(chain):
            if pair.key == key:
                deleted_pair = chain.pop(i)
                self.total_items -= 1
                print(f"✓ Deleted: {key} = {deleted_pair.value} (from index {index})")
                return True

        # Key not found
        print(f"X Key '{key}' not found in the hash table")
        return False

    def display(self):
```

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

[5] Resolving next ticket...
✓ Resolving: Ticket #101 | Customer: John Smith | Issue: Login issue
```

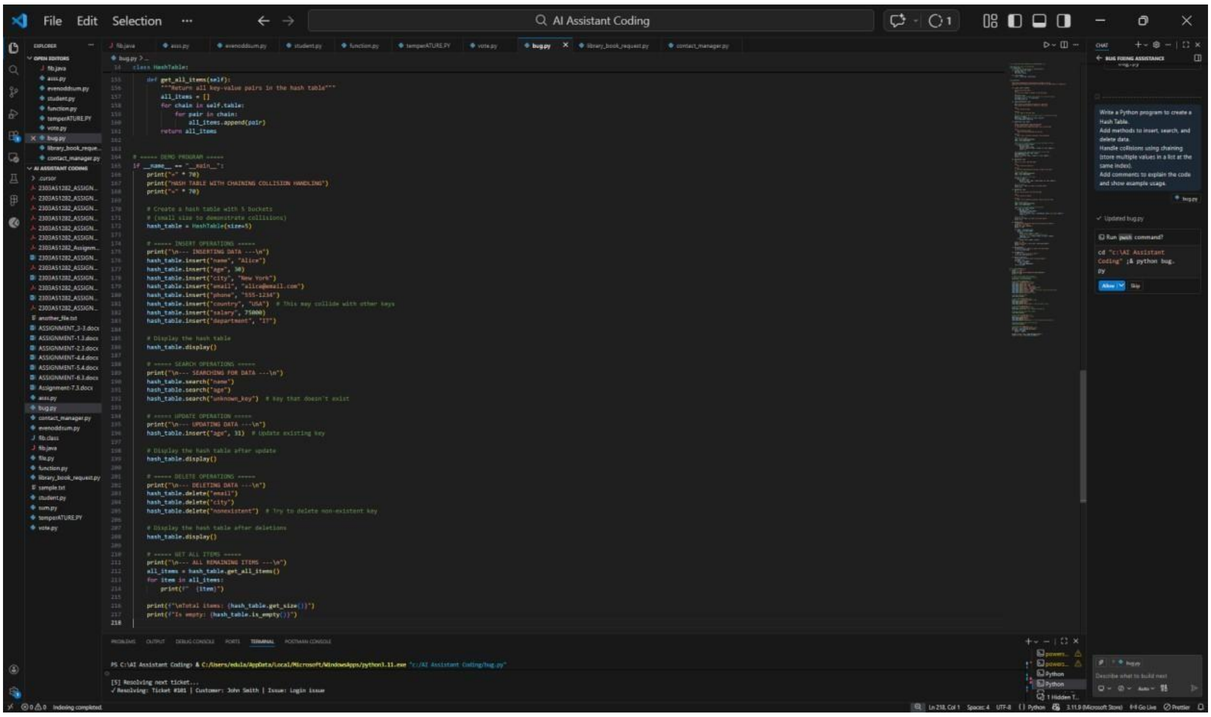
Write a Python program to create a Hash Table.
Add methods to insert, search, and delete data.
Handle collisions using chaining (store multiple values in a list at the same index).
Add comments to explain the code and show example usage.

✓ Updated bug.py

Run pwsh command?

```
cd "c:\AI Assistant Coding" ;& python bug.py
```

Allow | Skip



```python
class HashTable:
    def get_all_items(self):
        """Return 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("=" * 70)
    print("HASH TABLE WITH CHAINING COLLISION HANDLING")
    print("=" * 70)

    # Create a hash table with 5 buckets
    # (small size to demonstrate collisions)
    hash_table = HashTable(size=5)

    # ===== INSERT OPERATIONS =====
    print("\n--- INSERTING DATA ---\n")
    hash_table.insert("name", "Alice")
    hash_table.insert("age", 30)
    hash_table.insert("city", "New York")
    hash_table.insert("email", "alice@email.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("department", "IT")

    # Display the hash table
    hash_table.display()

    # ===== SEARCH OPERATIONS =====
    print("\n--- SEARCHING FOR DATA ---\n")
    hash_table.search("name")
    hash_table.search("age")
    hash_table.search("unknown_key")  # Key that doesn't exist

    # ===== UPDATE OPERATION =====
    print("\n--- UPDATING DATA ---\n")
    hash_table.insert("age", 31)  # Update existing key

    # Display the hash table after update
    hash_table.display()

    # ===== DELETE OPERATIONS =====
    print("\n--- DELETING DATA ---\n")
    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 REMAINING ITEMS ---\n")
    all_items = hash_table.get_all_items()
    for item in all_items:
        print(f"  {item}")

    print(f"\nTotal items: {hash_table.get_size()}")
    print(f"Is empty: {hash_table.is_empty()}")
```

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

[5] Resolving next ticket...
✓ Resolving: Ticket #101 | Customer: John Smith | Issue: Login issue
```

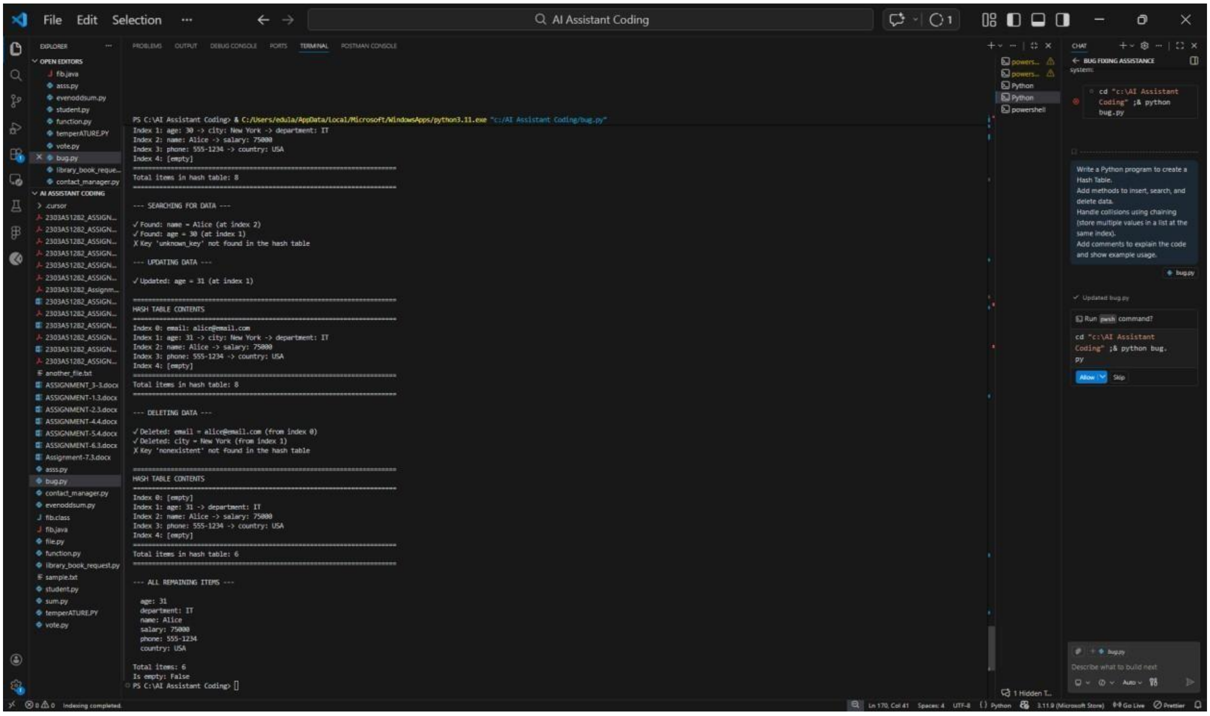
Write a Python program to create a Hash Table.
Add methods to insert, search, and delete data.
Handle collisions using chaining (store multiple values in a list at the same index).
Add comments to explain the code and show example usage.

Updated bug.py

Run pwsh command?

```
cd "c:\AI Assistant Coding" ;& python bug.py
```

Output:



```
PS C:\AI Assistant Coding> & C:/Users/edula/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/AI Assistant Coding/bug.py"
Index 1: age: 30 -> 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: 8
======================================================================

--- SEARCHING FOR DATA ---

✓ Found: name = Alice (at index 2)
✓ Found: age = 30 (at index 1)
X Key 'unknown_key' not found in the hash table

--- UPDATING DATA ---

✓ Updated: age = 31 (at index 1)

======================================================================
HASH TABLE CONTENTS
======================================================================
Index 0: email: alice@email.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: 8
======================================================================

--- DELETING DATA ---

✓ Deleted: email = alice@email.com (from index 0)
✓ Deleted: city = New York (from index 1)
X Key 'nonexistent' not found in the hash table

======================================================================
HASH 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: 6
======================================================================

--- ALL REMAINING ITEMS ---

  age: 31
  department: IT
  name: Alice
  salary: 75000
  phone: 555-1234
  country: USA

Total items: 6
Is empty: False
PS C:\AI Assistant Coding> 
```

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.

Updated bug.py

Run pwsh command?

```
cd "c:\AI Assistant Coding" ;& python bug.py
```

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

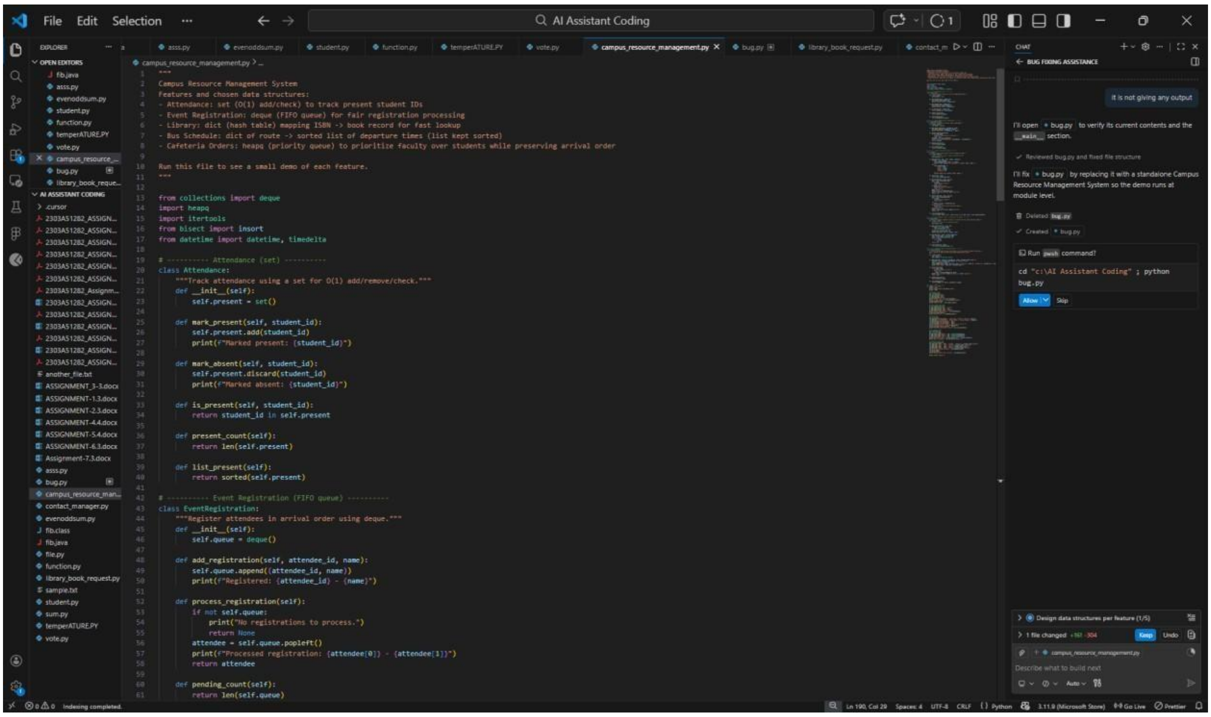
Design a Campus Resource Management System with the following features:

- 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

Code:



```python
"""
Campus Resource Management System
Features and chosen data structures:
- Attendance: set (O(1) add/check) to track present student IDs
- Event Registration: deque (FIFO queue) for fair registration processing
- Library: dict (hash table) mapping ISBN -> book record for fast lookup
- Bus Schedule: dict of route -> sorted list of departure times (list kept sorted)
- Cafeteria Orders: heapq (priority queue) to prioritize faculty over students while preserving arrival order

Run this file to see a small demo of each feature.
"""

from collections import deque
import heapq
import itertools
from bisect import insort
from datetime import datetime, timedelta

# ---------- Attendance (set) ----------
class Attendance:
    """Track attendance using a set for O(1) add/remove/check."""
    def __init__(self):
        self.present = set()

    def mark_present(self, student_id):
        self.present.add(student_id)
        print(f"Marked present: {student_id}")

    def mark_absent(self, student_id):
        self.present.discard(student_id)
        print(f"Marked absent: {student_id}")

    def is_present(self, student_id):
        return student_id in self.present

    def present_count(self):
        return len(self.present)

    def list_present(self):
        return sorted(self.present)

# ---------- Event Registration (FIFO queue) ----------
class EventRegistration:
    """Register attendees in arrival order using deque."""
    def __init__(self):
        self.queue = deque()

    def add_registration(self, attendee_id, name):
        self.queue.append((attendee_id, name))
        print(f"Registered: {attendee_id} - {name}")

    def process_registration(self):
        if not self.queue:
            print("No registrations to process.")
            return None
        attendee = self.queue.popleft()
        print(f"Processed registration: {attendee[0]} - {attendee[1]}")
        return attendee

    def pending_count(self):
        return len(self.queue)
```

BUG FIXING ASSISTANCE

it is not giving any output

I'll open bug.py to verify its current contents and the `__main__` section.

✓ Reviewed bug.py and fixed file structure

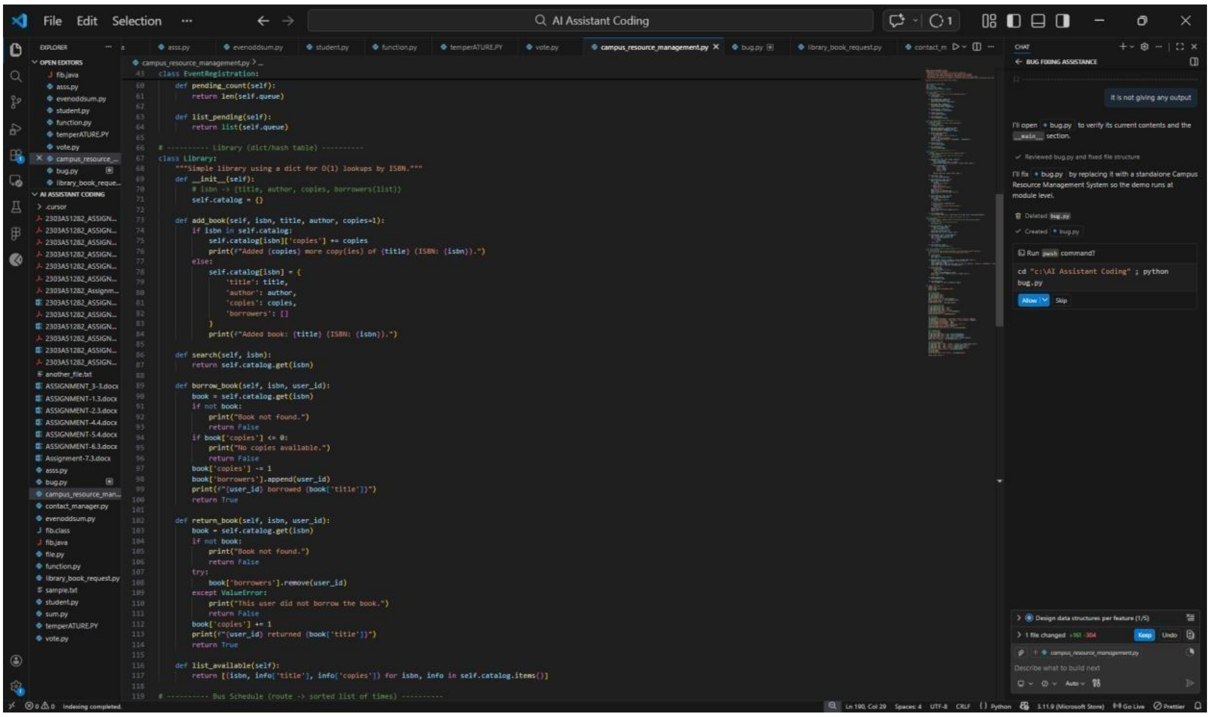
I'll fix bug.py by replacing it with a standalone Campus Resource Management System so the demo runs at module level.

Deleted bug.py

✓ Created bug.py

Run pwsh command?

```
cd "c:\AI Assistant Coding" ; python bug.py
```



```python
class EventRegistration:
    def pending_count(self):
        return len(self.queue)

    def list_pending(self):
        return list(self.queue)

# ---------- Library (dict/hash table) ----------
class Library:
    """Simple library using a dict for O(1) lookups by ISBN."""
    def __init__(self):
        # isbn -> {title, author, copies, borrowers(list)}
        self.catalog = {}

    def add_book(self, isbn, title, author, copies=1):
        if isbn in self.catalog:
            self.catalog[isbn]['copies'] += copies
            print(f"Added {copies} more copy(ies) of {title} (ISBN: {isbn}).")
        else:
            self.catalog[isbn] = {
                'title': title,
                'author': author,
                'copies': copies,
                'borrowers': []
            }
            print(f"Added book: {title} (ISBN: {isbn}).")

    def search(self, isbn):
        return self.catalog.get(isbn)

    def borrow_book(self, isbn, user_id):
        book = self.catalog.get(isbn)
        if not book:
            print("Book not found.")
            return False
        if book['copies'] <= 0:
            print("No copies available.")
            return False
        book['copies'] -= 1
        book['borrowers'].append(user_id)
        print(f"{user_id} borrowed {book['title']}")
        return True

    def return_book(self, isbn, user_id):
        book = self.catalog.get(isbn)
        if not book:
            print("Book not found.")
            return False
        try:
            book['borrowers'].remove(user_id)
        except ValueError:
            print("This user did not borrow the book.")
            return False
        book['copies'] += 1
        print(f"{user_id} returned {book['title']}")
        return True

    def list_available(self):
        return [(isbn, info['title'], info['copies']) for isbn, info in self.catalog.items()]

# ---------- Bus Schedule (route -> sorted list of times) ----------
```

BUG FIXING ASSISTANCE

It is not giving any output

I'll open bug.py to verify its current contents and the `__main__` section.

✓ Reviewed bug.py and fixed file structure

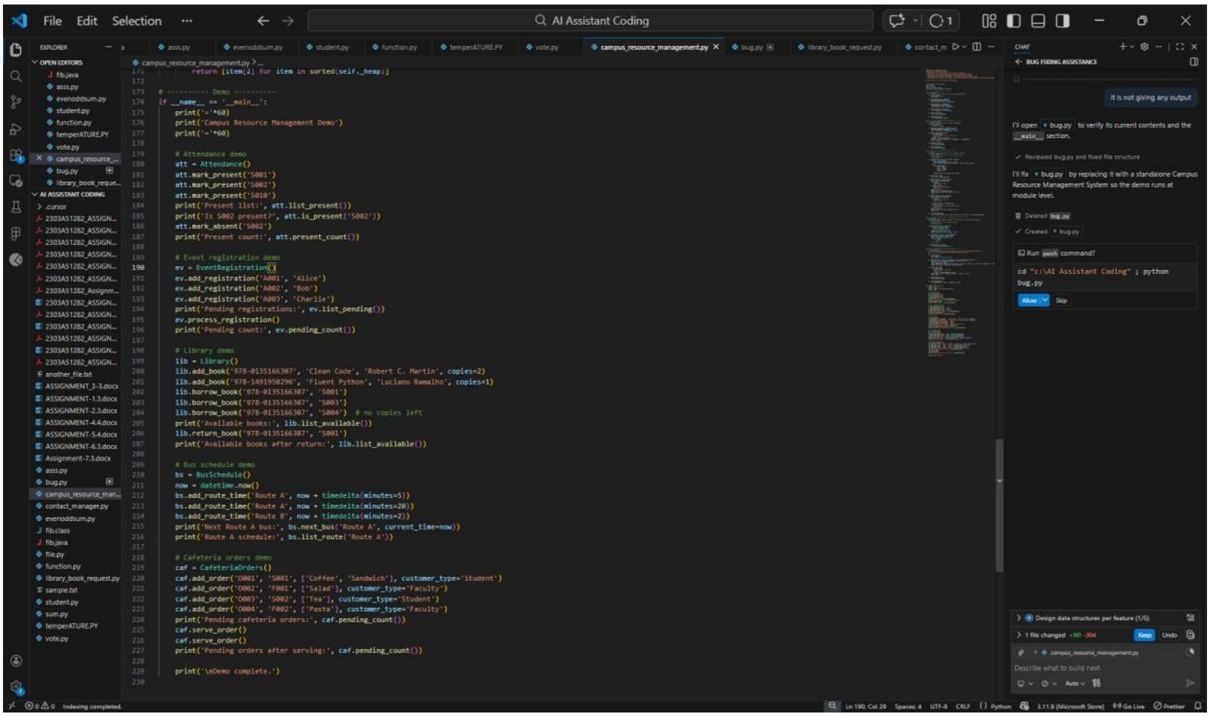
I'll fix bug.py by replacing it with a standalone Campus Resource Management System so the demo runs at module level.

Deleted bug.py

✓ Created bug.py

Run pwsh command?

```
cd "c:\AI Assistant Coding" ; python bug.py
```



```python
        return [item[2] for item in sorted(self._heap)]

# ---------- Demo ----------
if __name__ == '__main__':
    print('='*60)
    print('Campus Resource Management Demo')
    print('='*60)

    # Attendance demo
    att = Attendance()
    att.mark_present('S001')
    att.mark_present('S002')
    att.mark_present('S010')
    print('Present list:', att.list_present())
    print('Is S002 present?', att.is_present('S002'))
    att.mark_absent('S002')
    print('Present count:', att.present_count())

    # Event registration demo
    ev = EventRegistration()
    ev.add_registration('A001', 'Alice')
    ev.add_registration('A002', 'Bob')
    ev.add_registration('A003', 'Charlie')
    print('Pending registrations:', ev.list_pending())
    ev.process_registration()
    print('Pending count:', ev.pending_count())

    # Library demo
    lib = Library()
    lib.add_book('978-0135166307', 'Clean Code', 'Robert C. Martin', copies=2)
    lib.add_book('978-1491950296', 'Fluent Python', 'Luciano Ramalho', copies=1)
    lib.borrow_book('978-0135166307', 'S001')
    lib.borrow_book('978-0135166307', 'S003')
    lib.borrow_book('978-0135166307', 'S004')  # no copies left
    print('Available books:', lib.list_available())
    lib.return_book('978-0135166307', 'S001')
    print('Available books after return:', lib.list_available())

    # Bus schedule demo
    bs = BusSchedule()
    now = datetime.now()
    bs.add_route_time('Route A', now + timedelta(minutes=5))
    bs.add_route_time('Route A', now + timedelta(minutes=20))
    bs.add_route_time('Route B', now + timedelta(minutes=2))
    print('Next Route A bus:', bs.next_bus('Route A', current_time=now))
    print('Route A schedule:', bs.list_route('Route A'))

    # Cafeteria orders demo
    caf = CafeteriaOrders()
    caf.add_order('O001', 'S001', ['Coffee', 'Sandwich'], customer_type='Student')
    caf.add_order('O002', 'F001', ['Salad'], customer_type='Faculty')
    caf.add_order('O003', 'S002', ['Tea'], customer_type='Student')
    caf.add_order('O004', 'F002', ['Pasta'], customer_type='Faculty')
    print('Pending cafeteria orders:', caf.pending_count())
    caf.serve_order()
    caf.serve_order()
    print('Pending orders after serving:', caf.pending_count())

    print('\nDemo complete.')
```

BUG FIXING ASSISTANCE

It is not giving any output

I'll open bug.py to verify its current contents and the `__main__` section.

✓ Reviewed bug.py and fixed file structure

I'll fix bug.py by replacing it with a standalone Campus Resource Management System so the demo runs at module level.

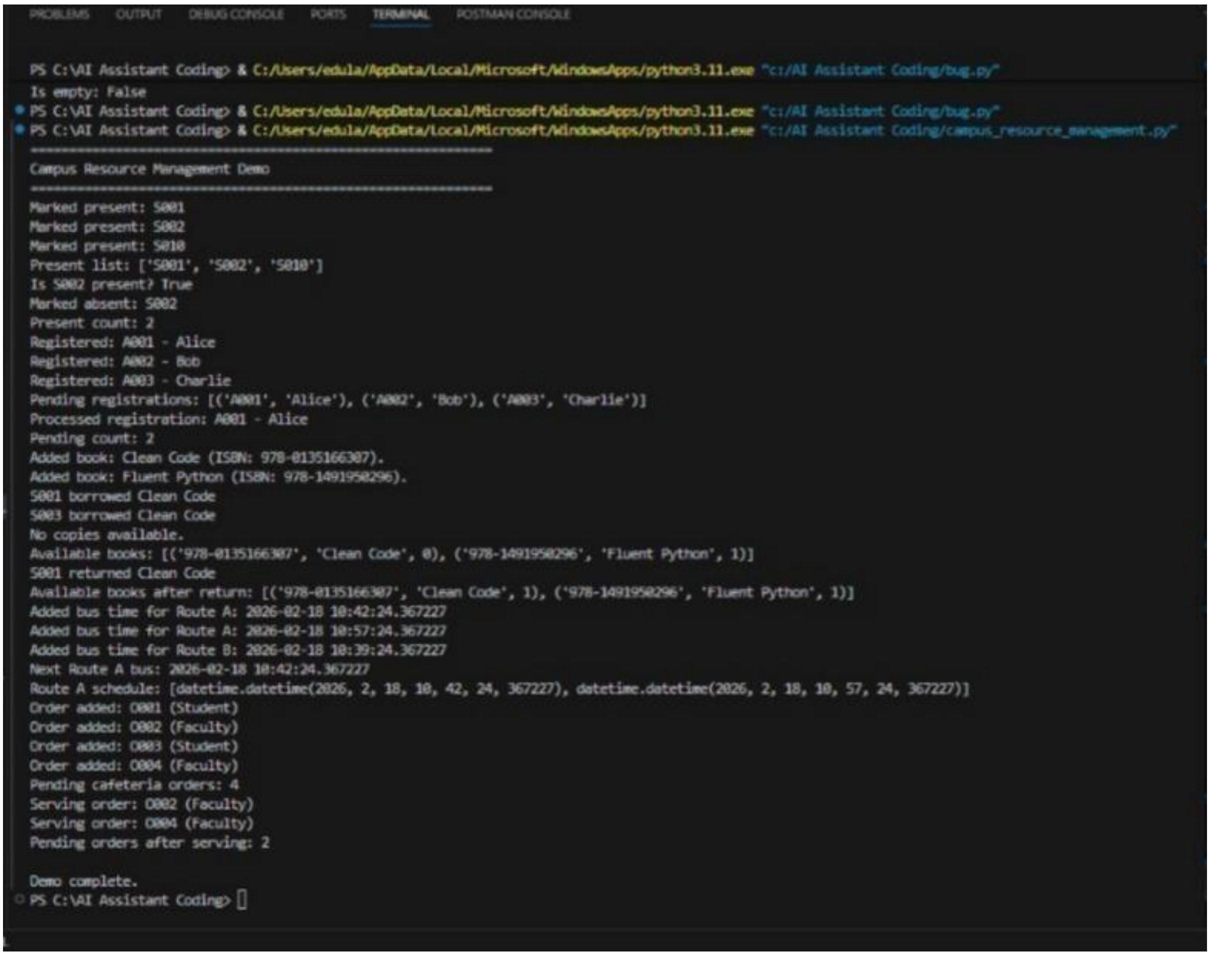
Deleted bug.py

✓ Created bug.py

Run pwsh command?

cd "c:\AI Assistant Coding" ; python bug.py

Output:



```
PS C:\AI Assistant Coding> & C:/Users/edula/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/AI Assistant Coding/bug.py"
Is empty: False
PS C:\AI Assistant Coding> & C:/Users/edula/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/AI Assistant Coding/bug.py"
PS C:\AI Assistant Coding> & C:/Users/edula/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/AI Assistant Coding/campus_resource_management.py"
============================================================
Campus Resource Management Demo
============================================================
Marked present: S001
Marked present: S002
Marked present: S010
Present list: ['S001', 'S002', 'S010']
Is S002 present? True
Marked absent: S002
Present count: 2
Registered: A001 - Alice
Registered: A002 - Bob
Registered: A003 - Charlie
Pending registrations: [('A001', 'Alice'), ('A002', 'Bob'), ('A003', 'Charlie')]
Processed registration: A001 - Alice
Pending count: 2
Added book: Clean Code (ISBN: 978-0135166307).
Added book: Fluent Python (ISBN: 978-1491950296).
S001 borrowed Clean Code
S003 borrowed Clean Code
No copies available.
Available books: [('978-0135166307', 'Clean Code', 0), ('978-1491950296', 'Fluent Python', 1)]
S001 returned Clean Code
Available books after return: [('978-0135166307', 'Clean Code', 1), ('978-1491950296', 'Fluent Python', 1)]
Added bus time for Route A: 2026-02-18 10:42:24.367227
Added bus time for Route A: 2026-02-18 10:57:24.367227
Added bus time for Route B: 2026-02-18 10:39:24.367227
Next Route A bus: 2026-02-18 10:42:24.367227
Route A schedule: [datetime.datetime(2026, 2, 18, 10, 42, 24, 367227), datetime.datetime(2026, 2, 18, 10, 57, 24, 367227)]
Order added: O001 (Student)
Order added: O002 (Faculty)
Order added: O003 (Student)
Order added: O004 (Faculty)
Pending cafeteria orders: 4
Serving order: O002 (Faculty)
Serving order: O004 (Faculty)
Pending orders after serving: 2

Demo complete.
PS C:\AI Assistant Coding> 
```

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.