ASSIGNMENT-11.3

Name: E.Ramya

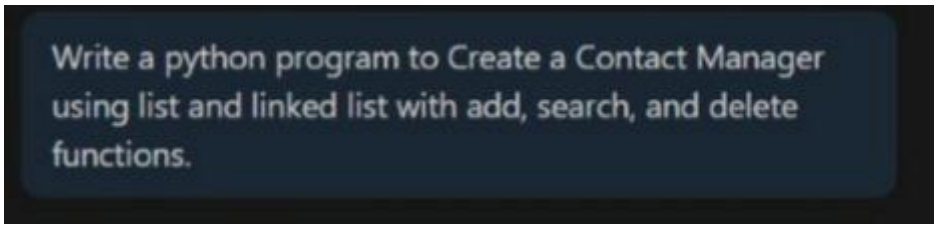
Ht.no: 2303A51282

Batch: 05

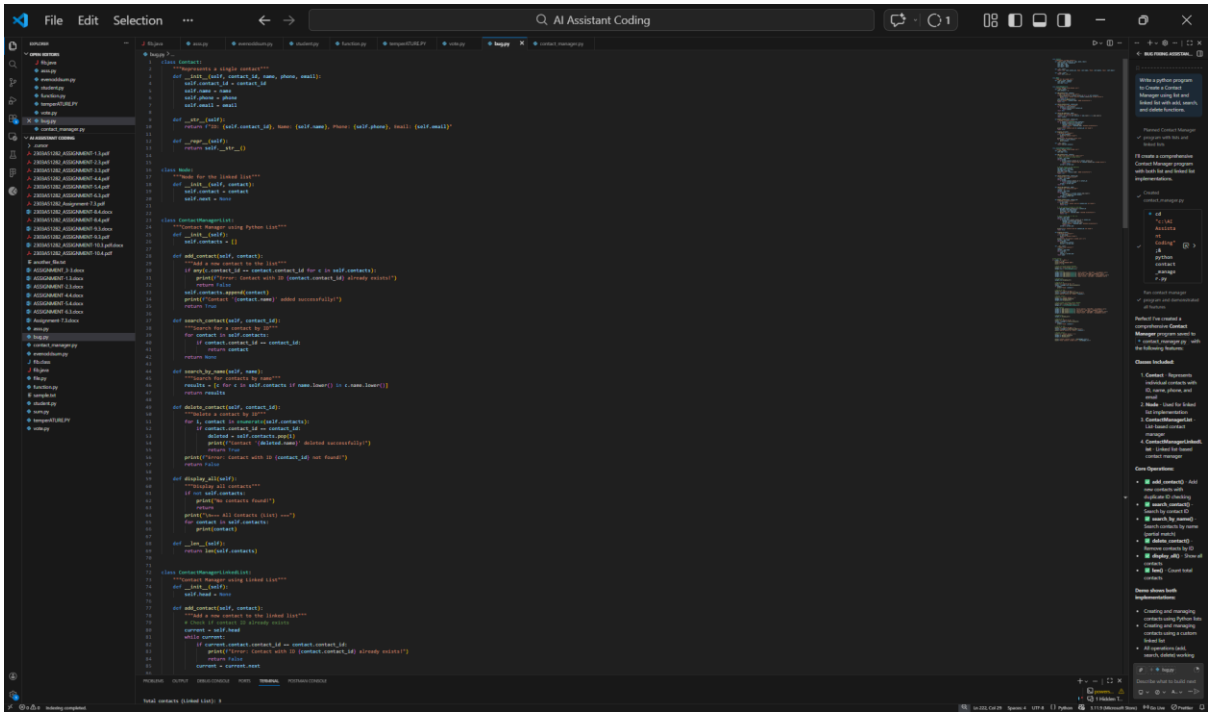
Task 1: Smart Contact Manager (Arrays & Linked Lists)
Scenario
SR University's student club requires a simple Contact Manager Application to store members' names and phone numbers. The system should support efficient addition, searching, and deletion of contacts.

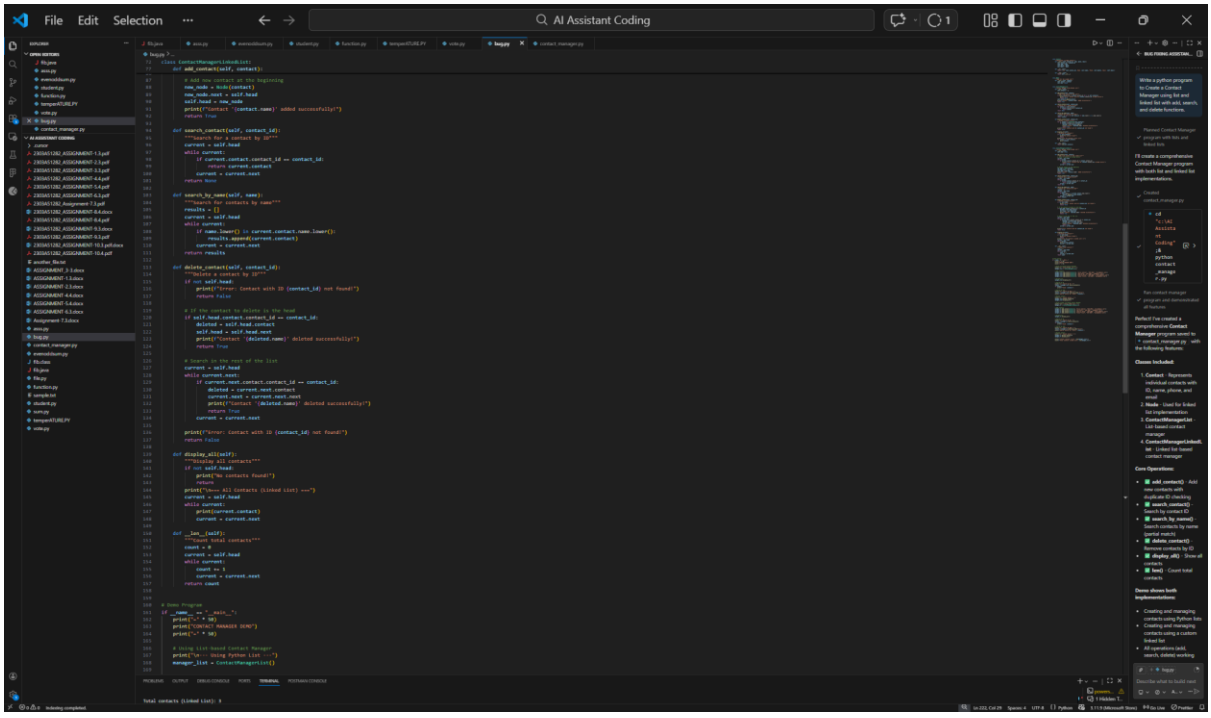
Prompt:

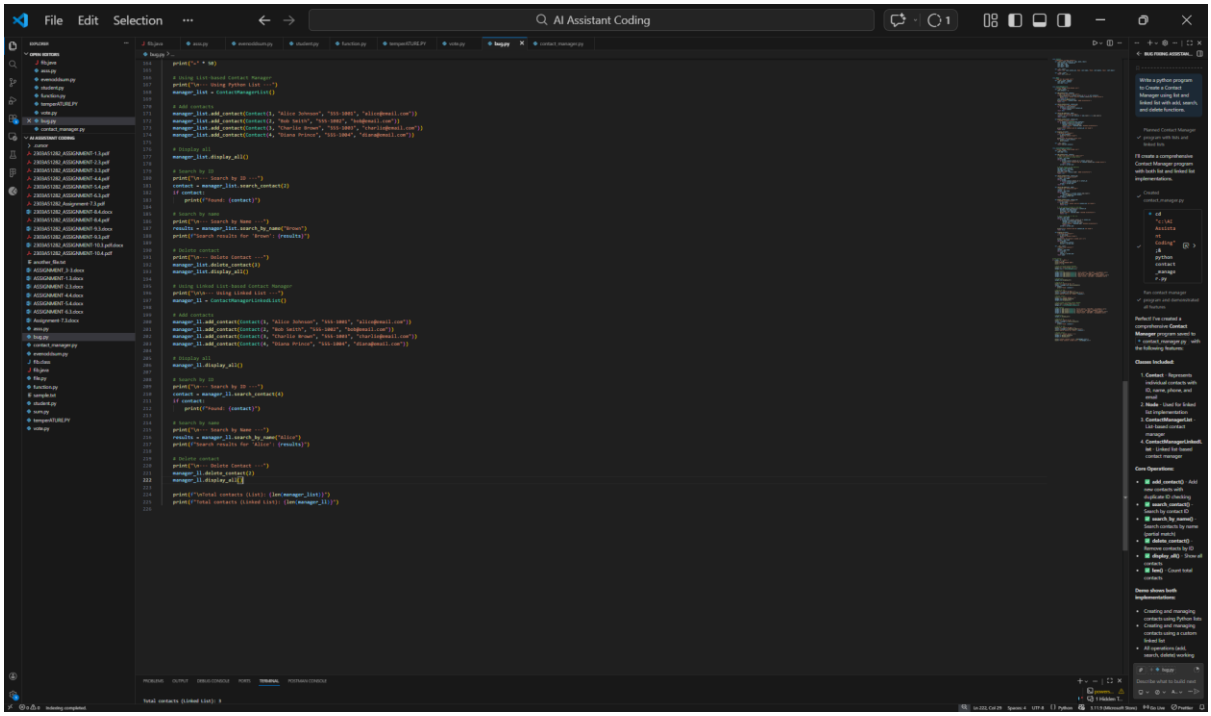
Write a python program to Create a Contact Manager using list and linked list with add, search, and delete functions.



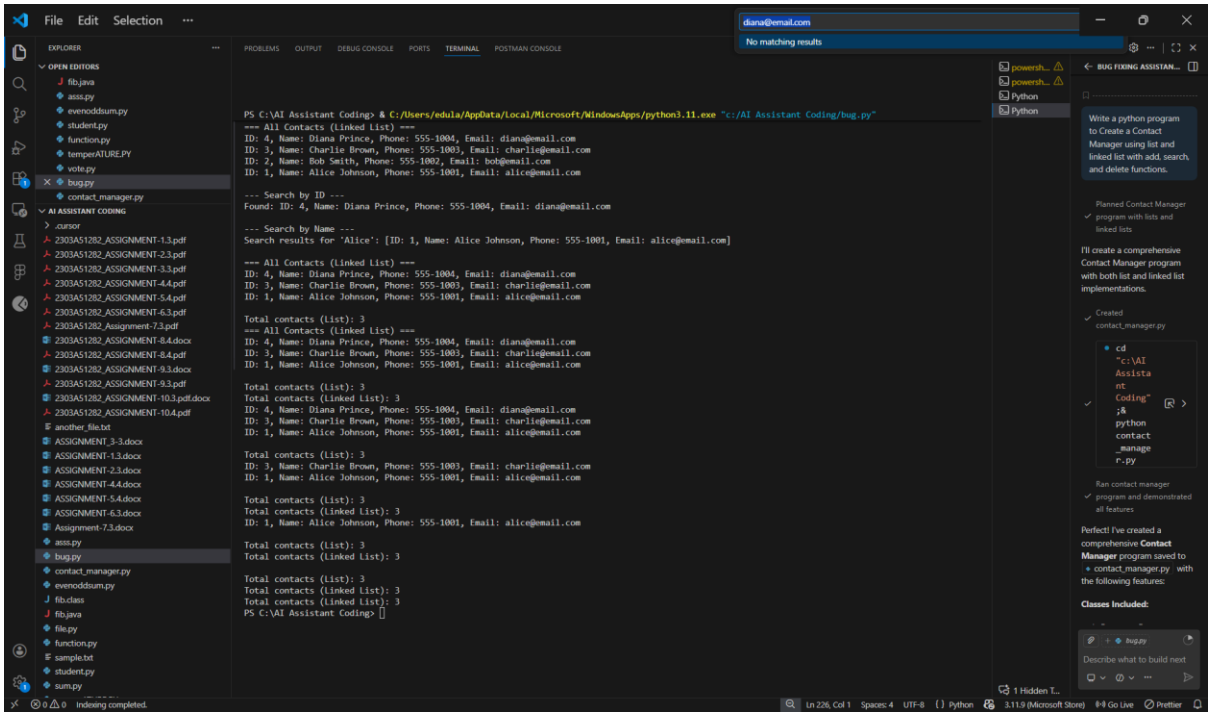
Code:


File Edit Selection
AI Assistant Coding


File Edit Selection
AI Assistant Coding



Output:



Explanation:

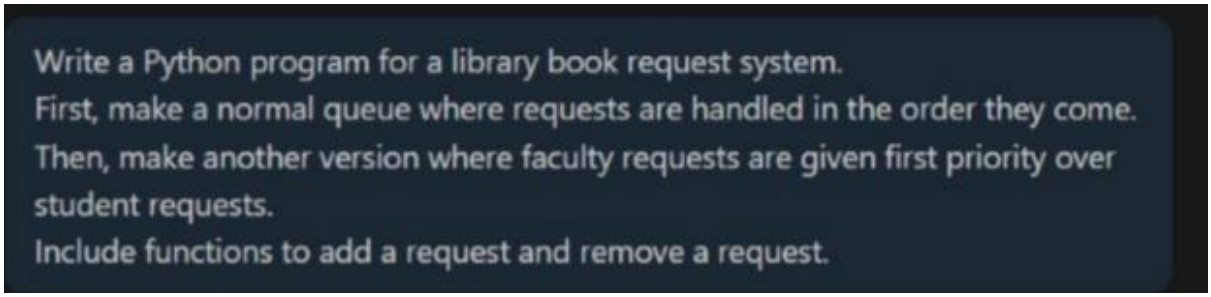
- In an array, adding at the end is fast, but inserting in the middle is slow because elements must shift.
- In a linked list, insertion is fast because no shifting is needed.
- Searching takes the same time in both (you must check each element).
- Deleting in an array is slower due to shifting elements.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

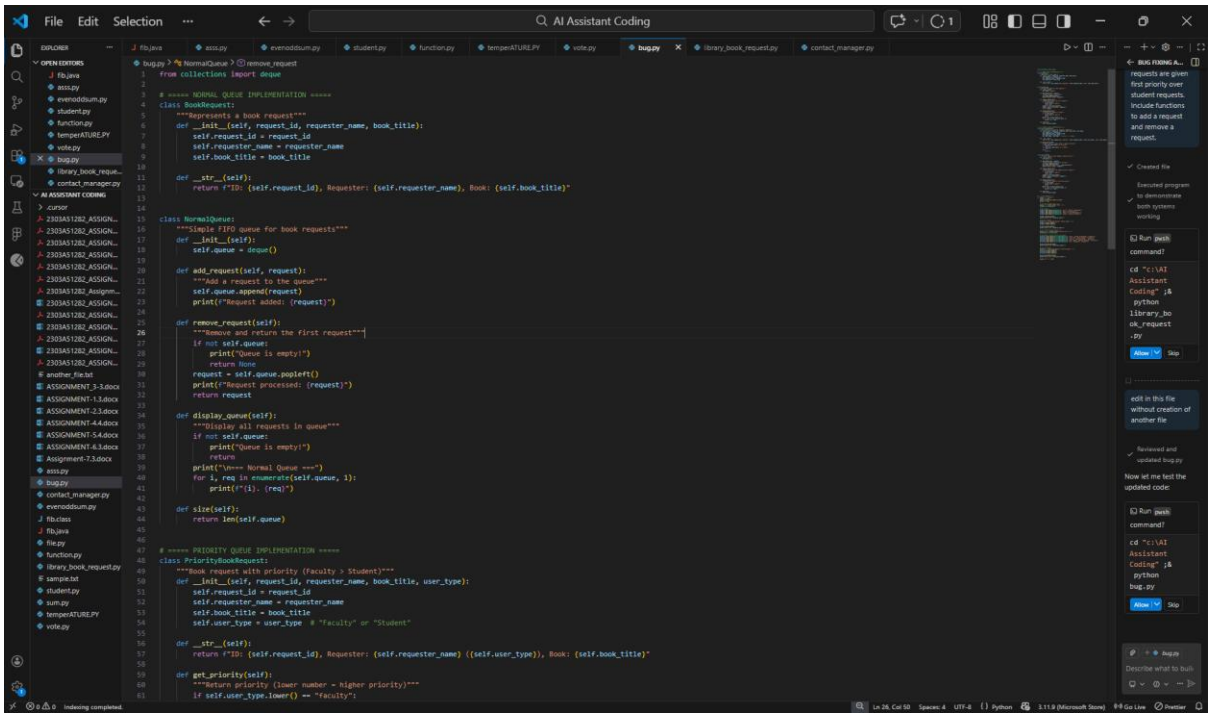
Prompt:

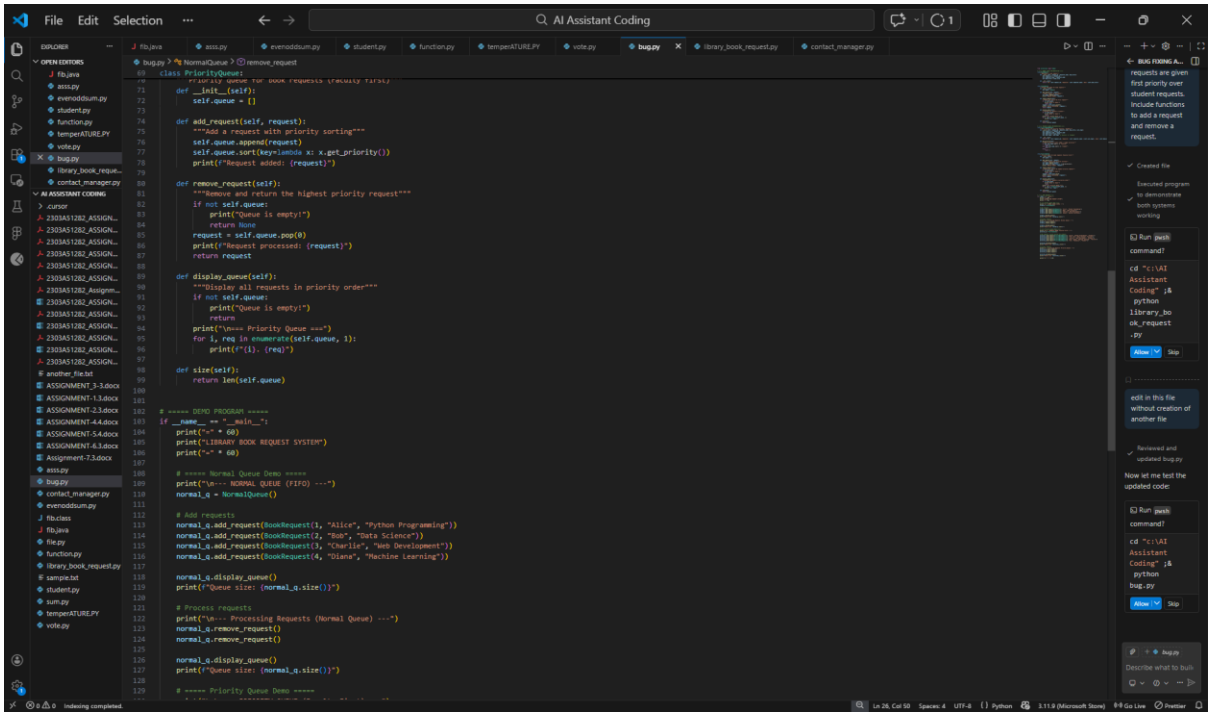
Write a Python program for a library book request system.First, make a normal queue where requests are handled in the order they come.Then, make another version where faculty requests are given

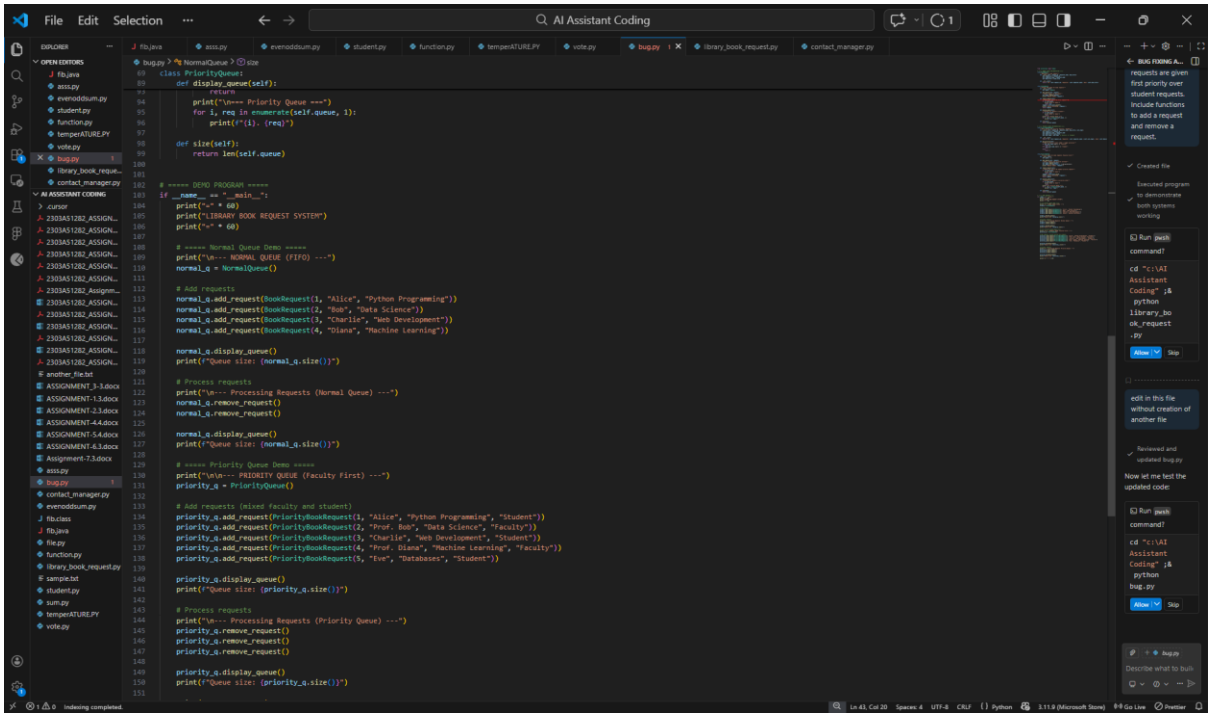
first priority over student requests.Include functions to add a request and remove a request.



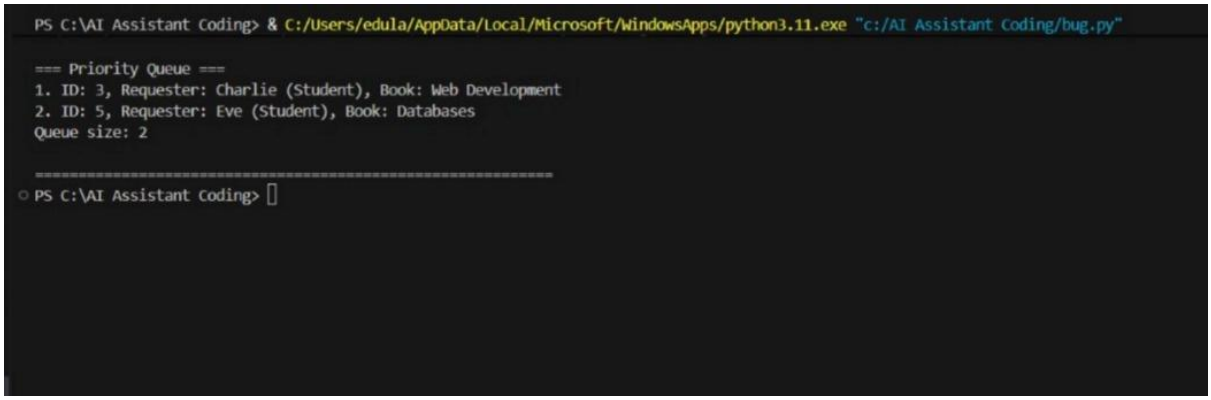
Code:




File Edit Selection
AI Assistant Coding
bug.py
class PriorityQueue:
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 =====


File Edit Selection
AI Assistant Coding
bug.py
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--- 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()}")

Output:



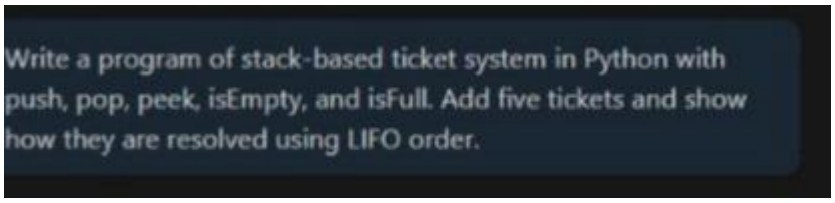
Explanation:

- 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.

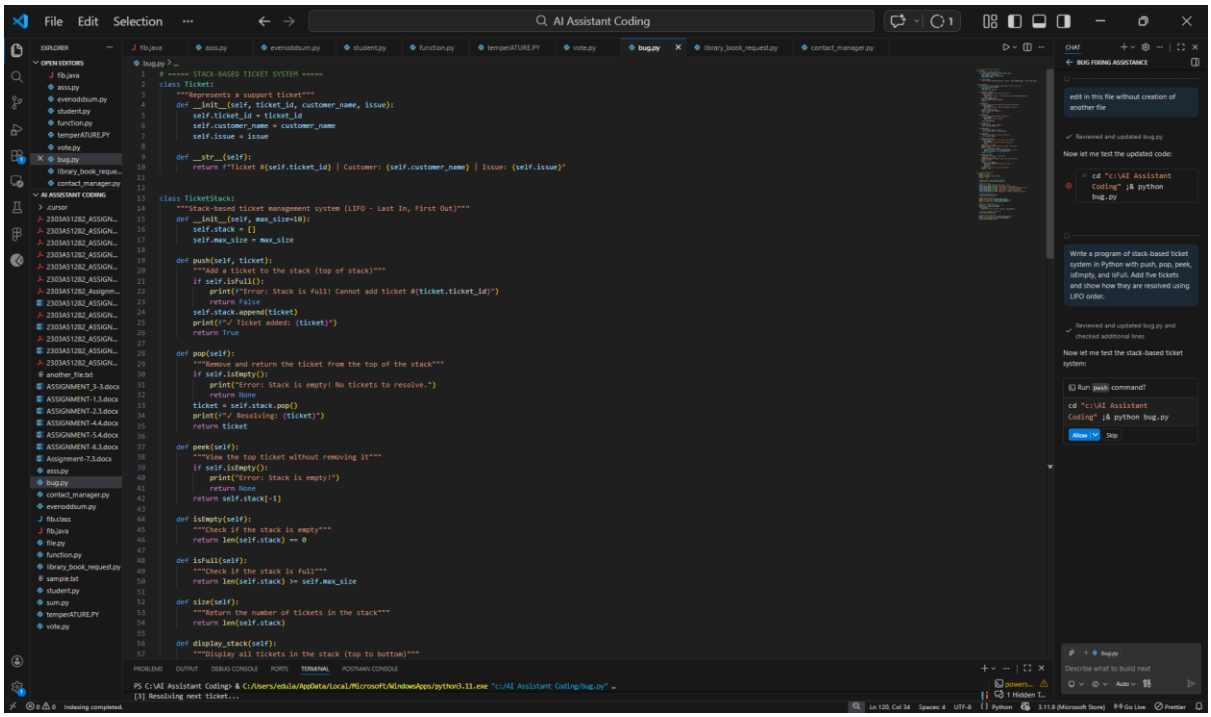
Task 3: Emergency Help Desk (Stack Implementation)
Scenario
SR University's IT Help Desk receives technical support tickets from students and staff. While tickets are received sequentially, issue escalation follows a Last-In, First-Out (LIFO) approach.

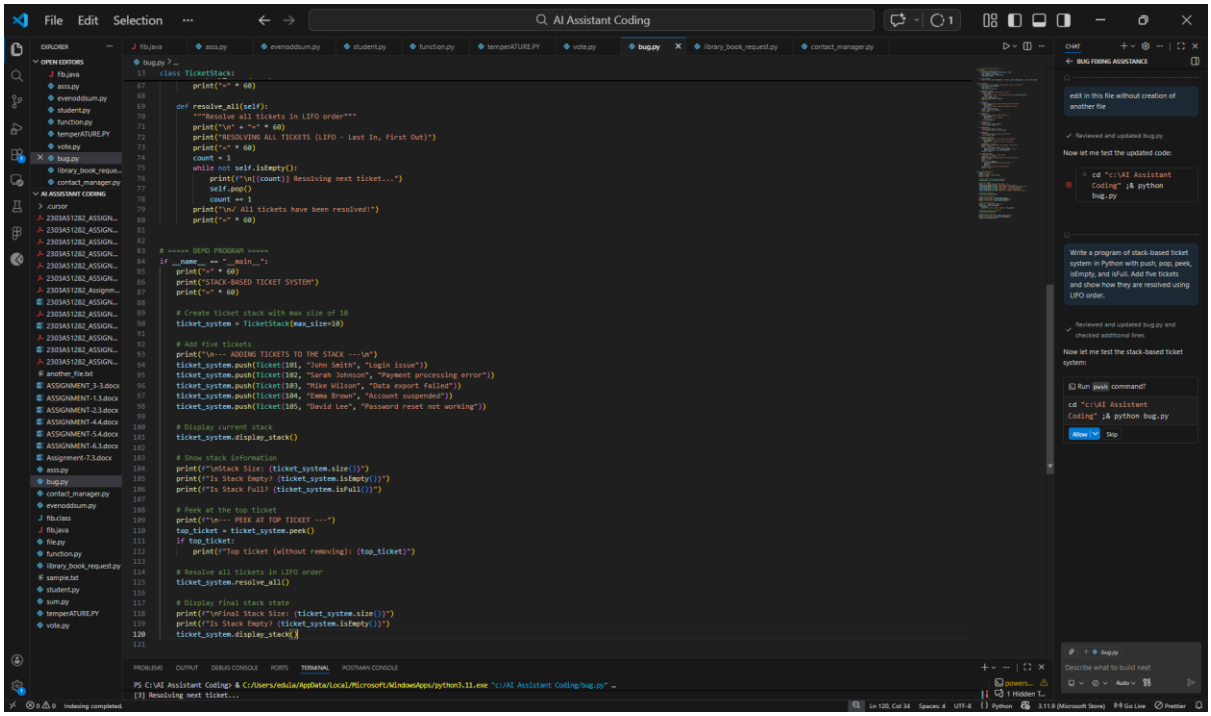
Prompt:

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.

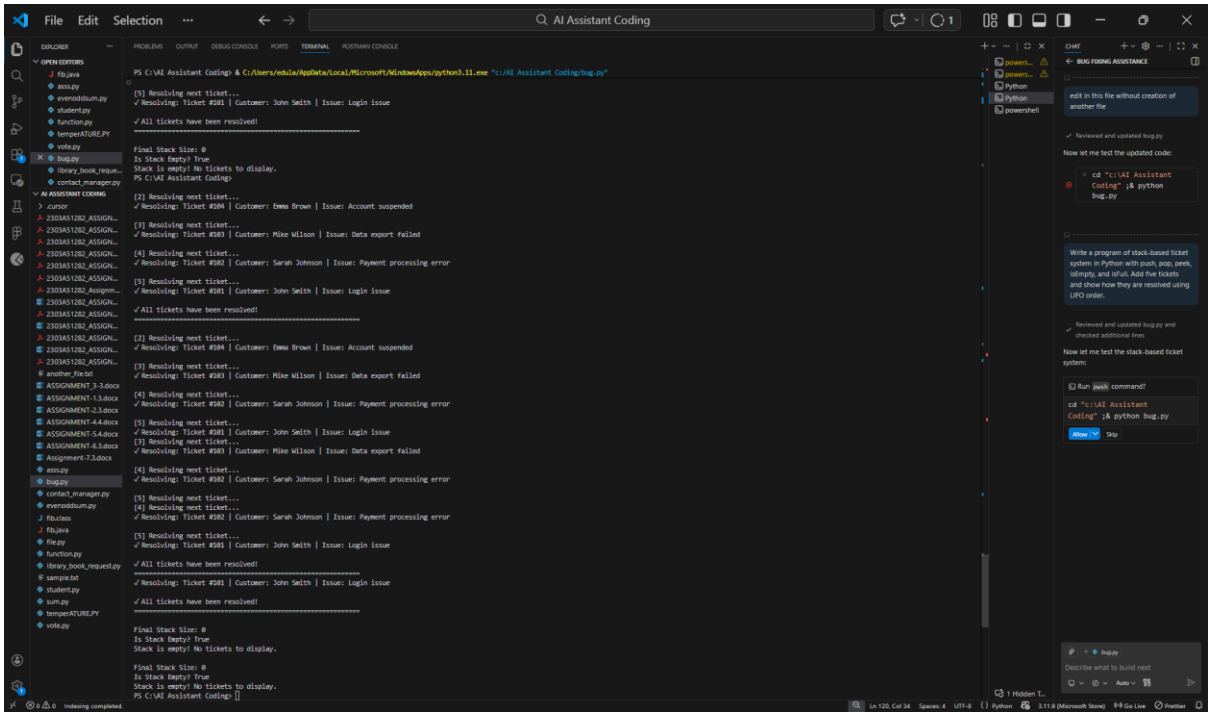


Code:





Output:



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.

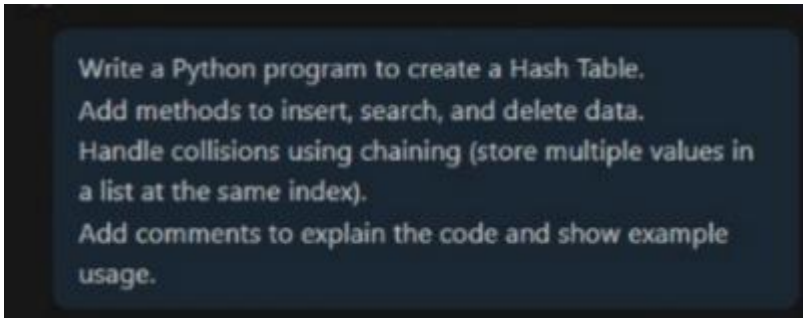
Task 4: Hash Table

Objective

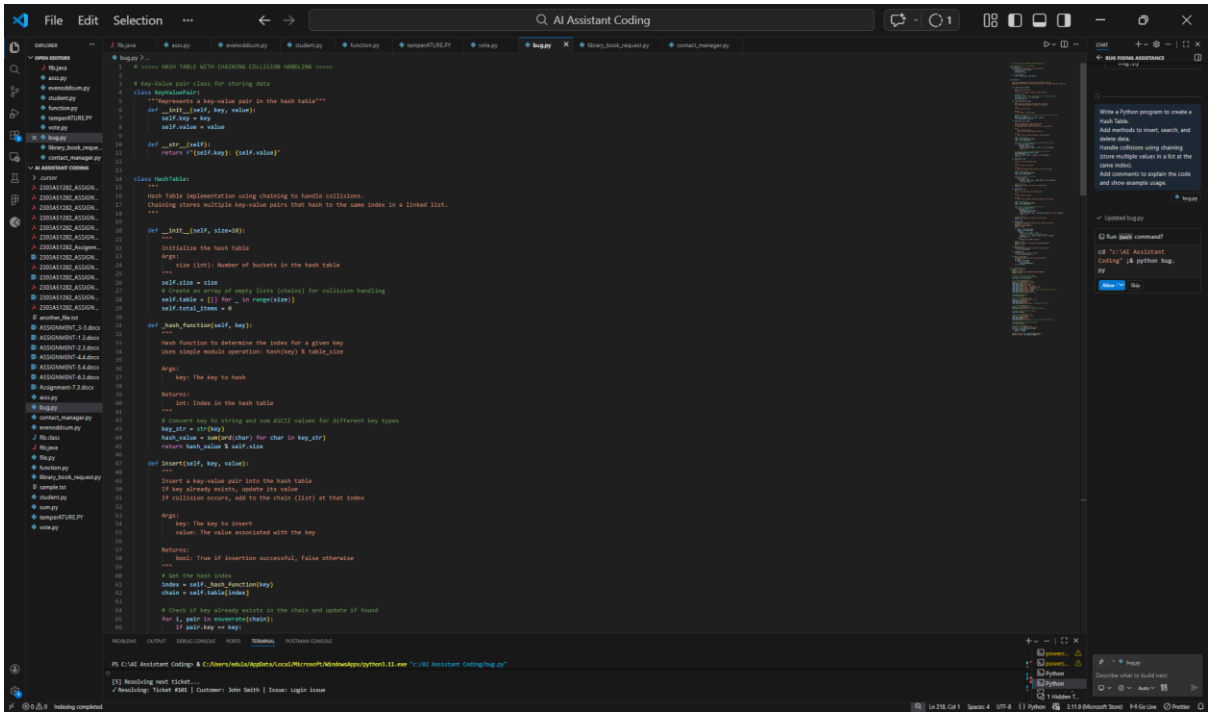
To implement a Hash Table and understand collision handling.

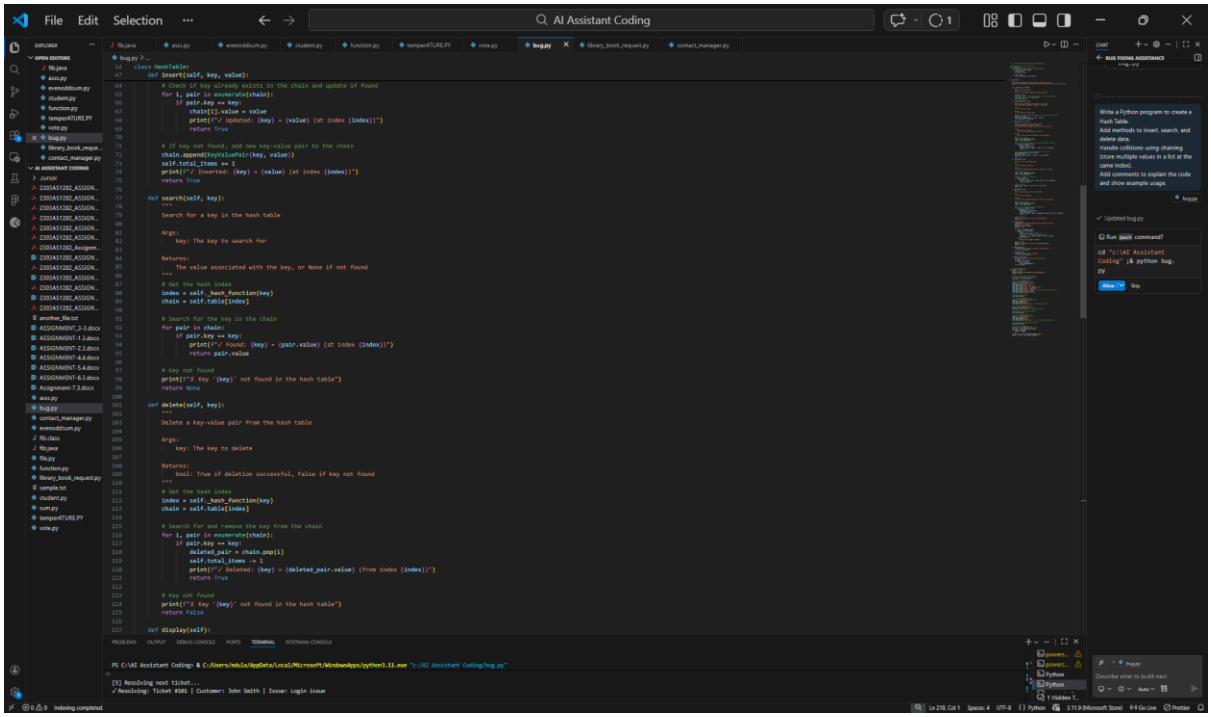
Prompt:

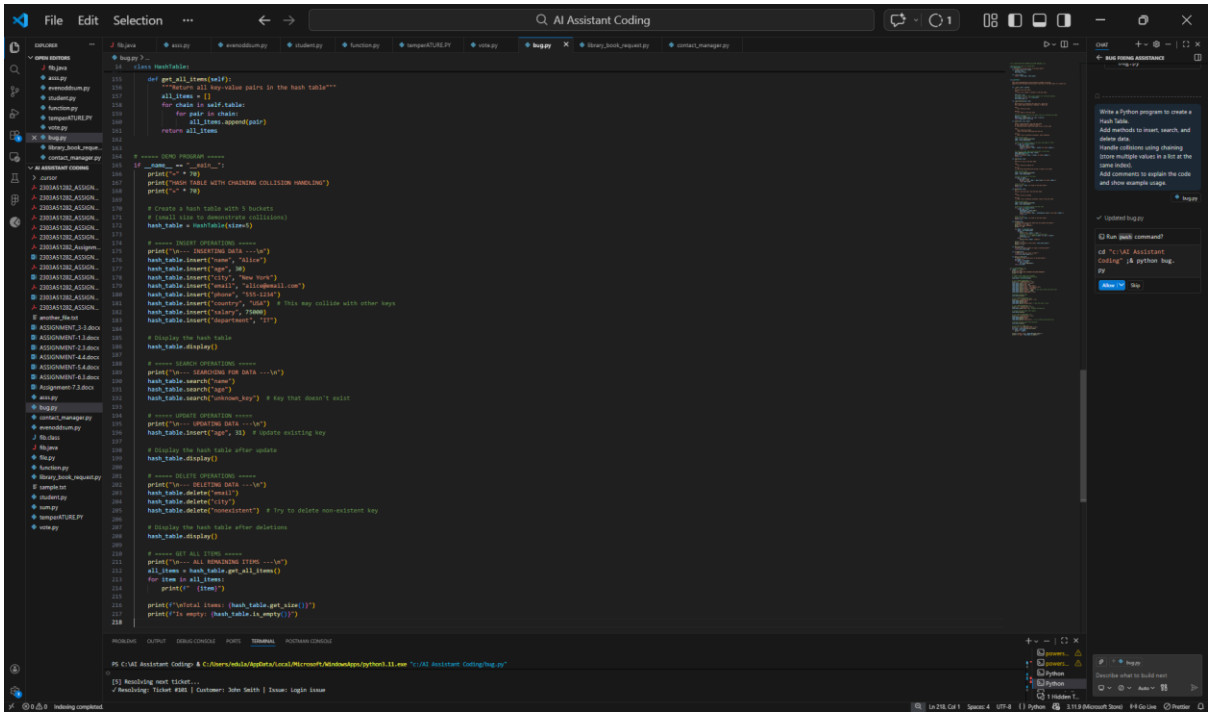
Write a Python program to create a Hash Table.
Add methods to insert, search, and delete data.
Handle collisions using chaining (store multiple values in a list at the same index).
Add comments to explain the code and show example usage.



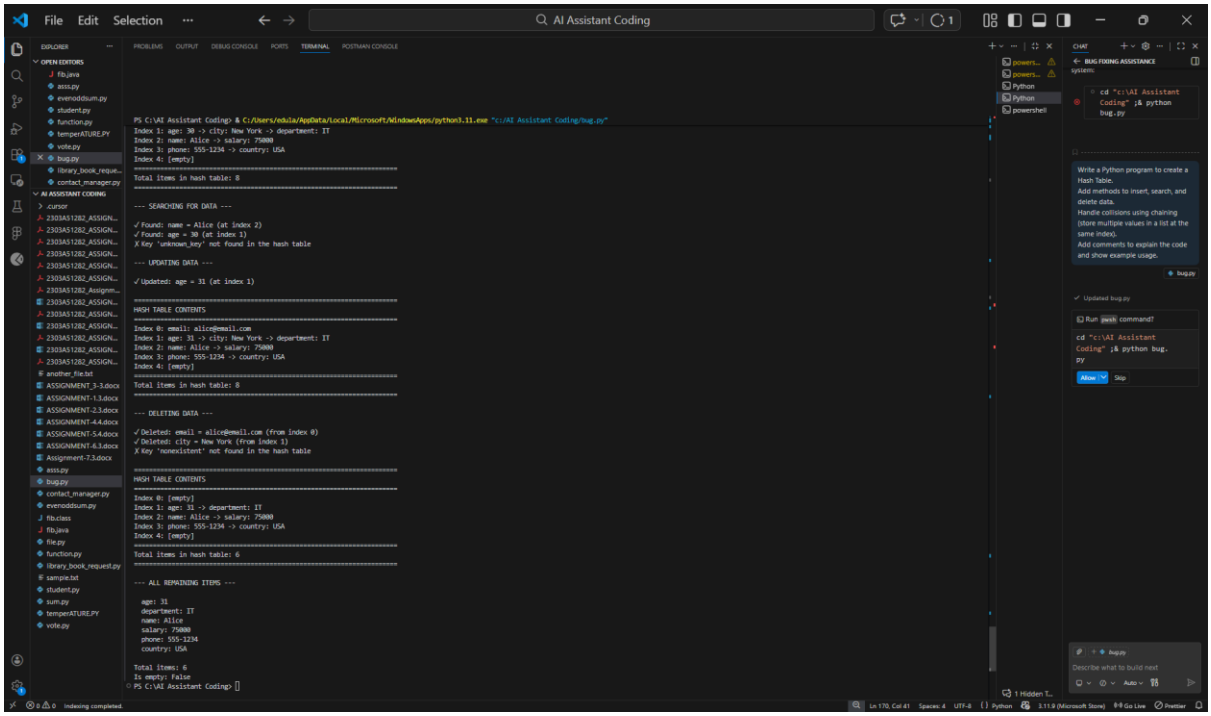
Code:







Output:



Explanation:

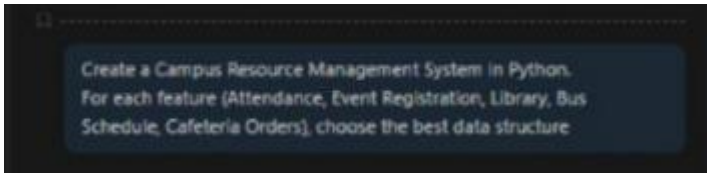
- A Hash Table stores data using a key and value.
- A hash function decides where to store the data.

- 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.

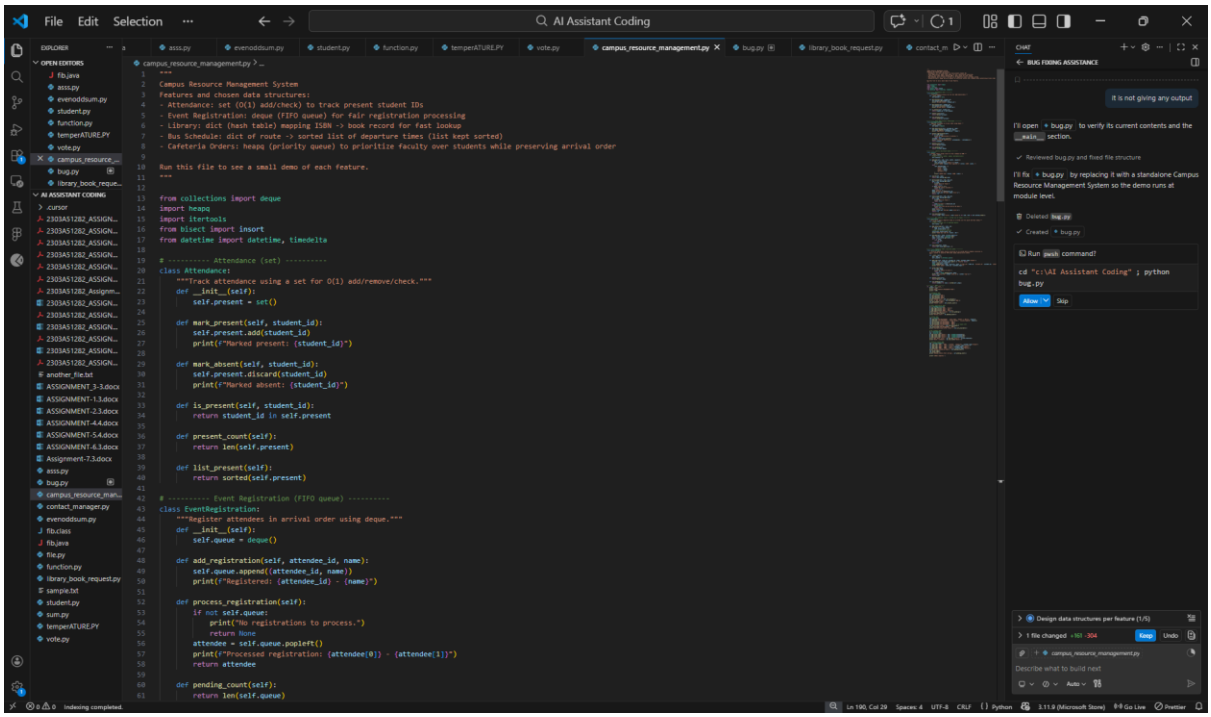
Task 5: Real-Time Application Challenge
Scenario
Design a Campus Resource Management System with the following
features:
• Student Attendance Tracking
• Event Registration System
• Library Book Borrowing
• Bus Scheduling System
• Cafeteria Order Queue

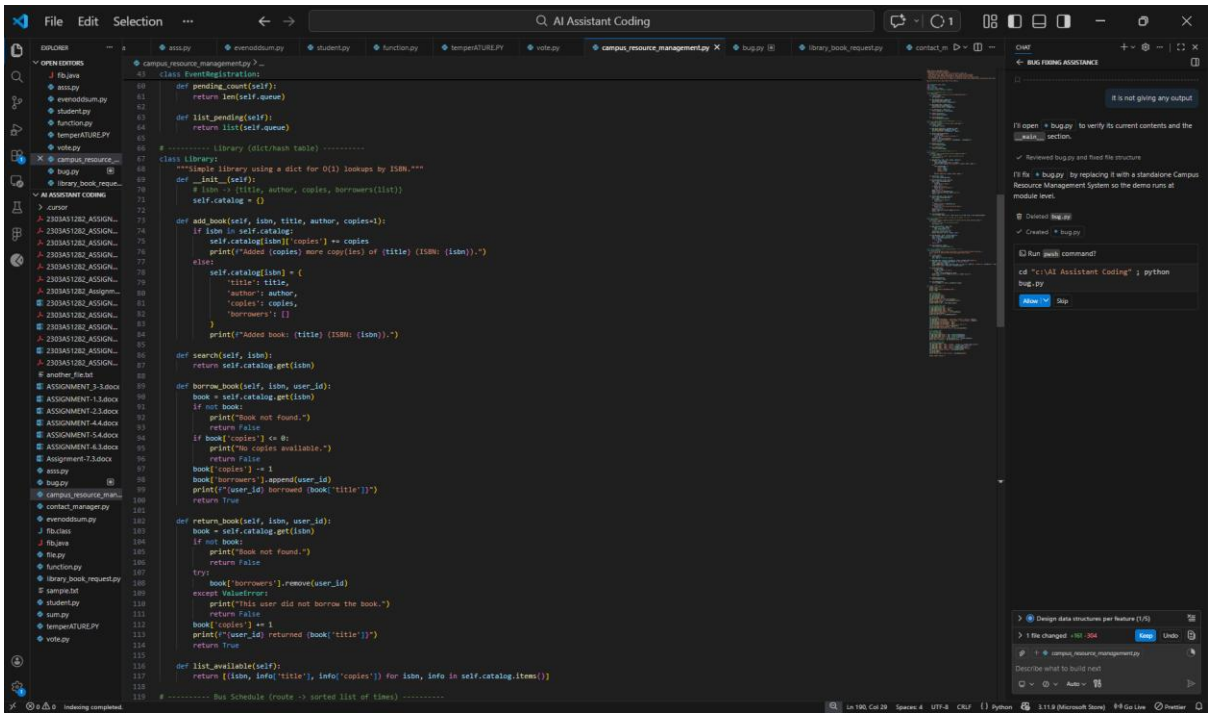
Prompt:

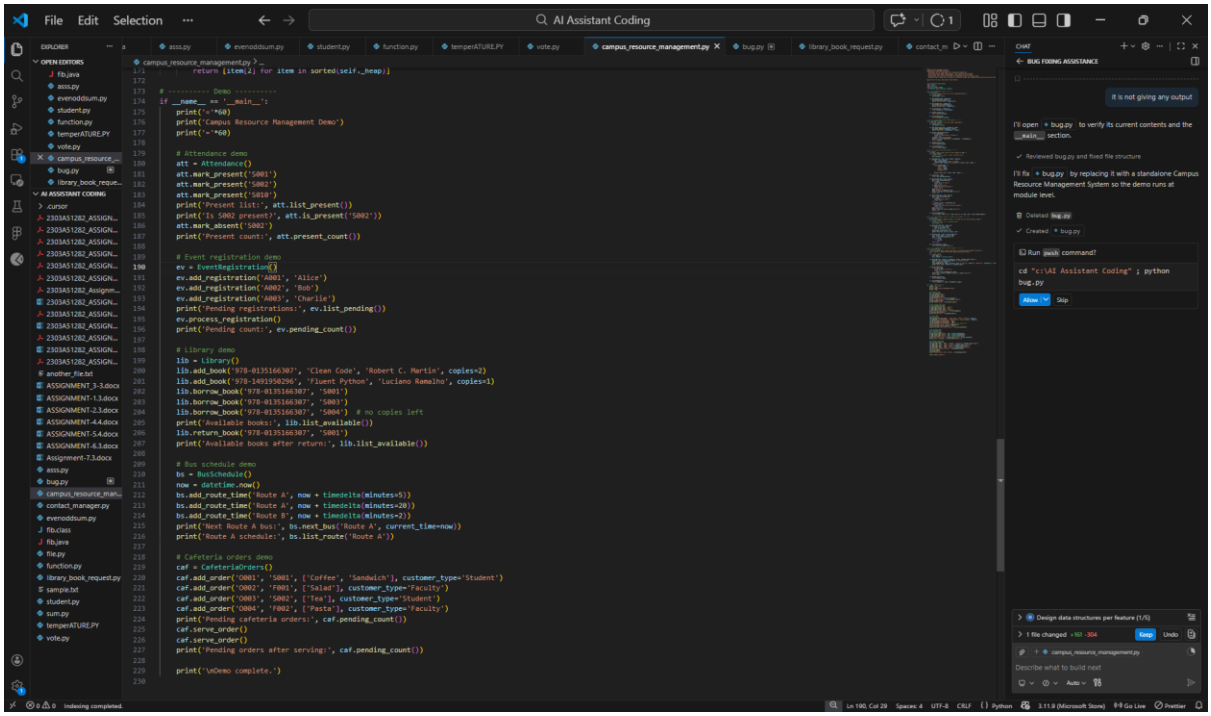
Create a Campus Resource Management System in Python.For each feature (Attendance, Event Registration, Library, Bus Schedule, Cafeteria Orders), choose the best data structure


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

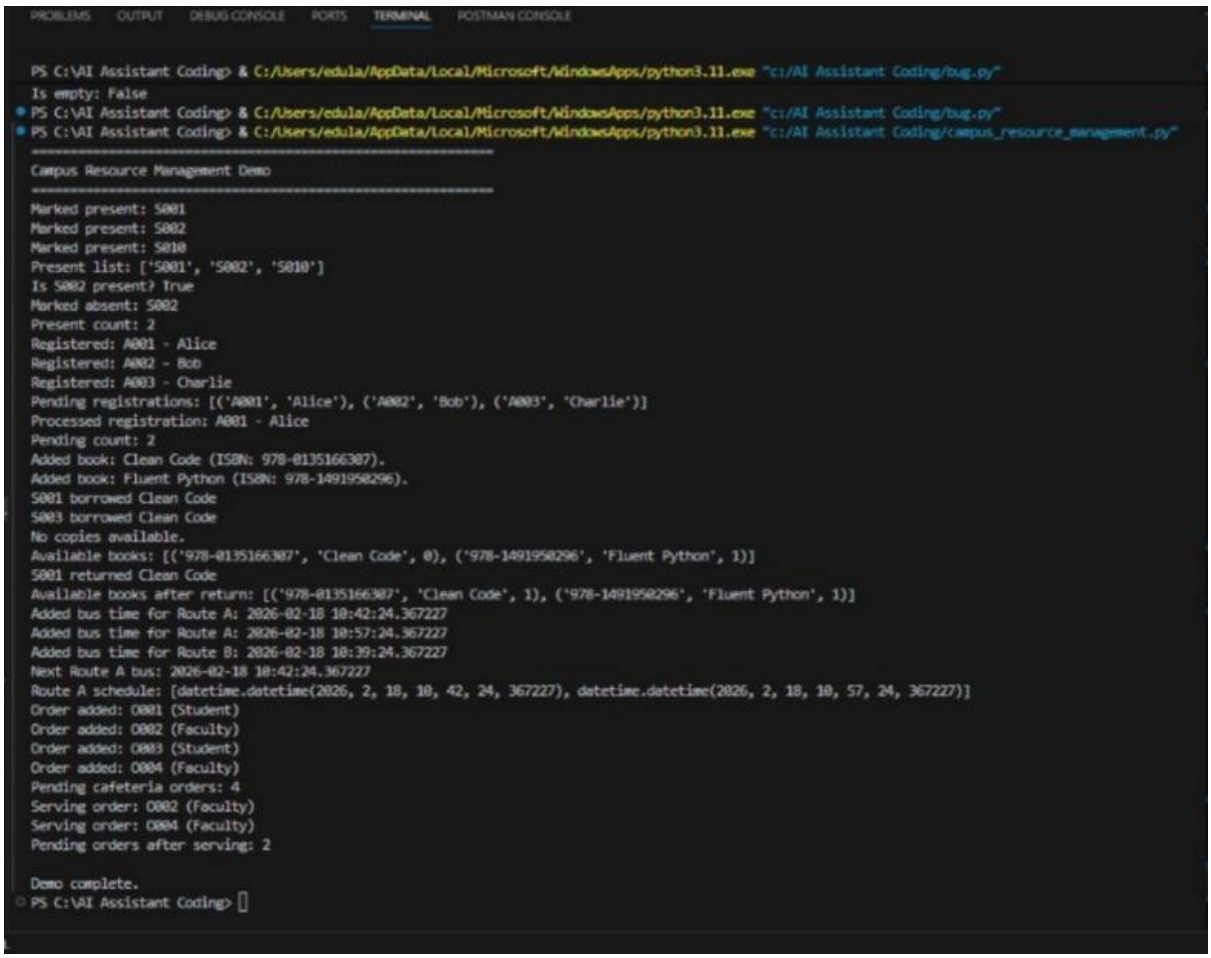
Code:


File Edit Selection
AI Assistant Coding
campus_resource_management.py
BUG FIXING ASSISTANCE
it is not giving any output


File Edit Selection
AI Assistant Coding
campus_resource_management.py
BUG FIXING ASSISTANCE
it is not giving any output


File Edit Selection
AI Assistant Coding
campus_resource_management.py
BUG FIXING ASSISTANCE
it is not giving any output

Output:


PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL POSTMAN CONSOLE
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.