

Assignment 11.3 Ai Assisted Coding

Ht.no: 2303A510C1

Batch: 06

Task 1:

Smart Contact Manager (Arrays & Linked Lists)

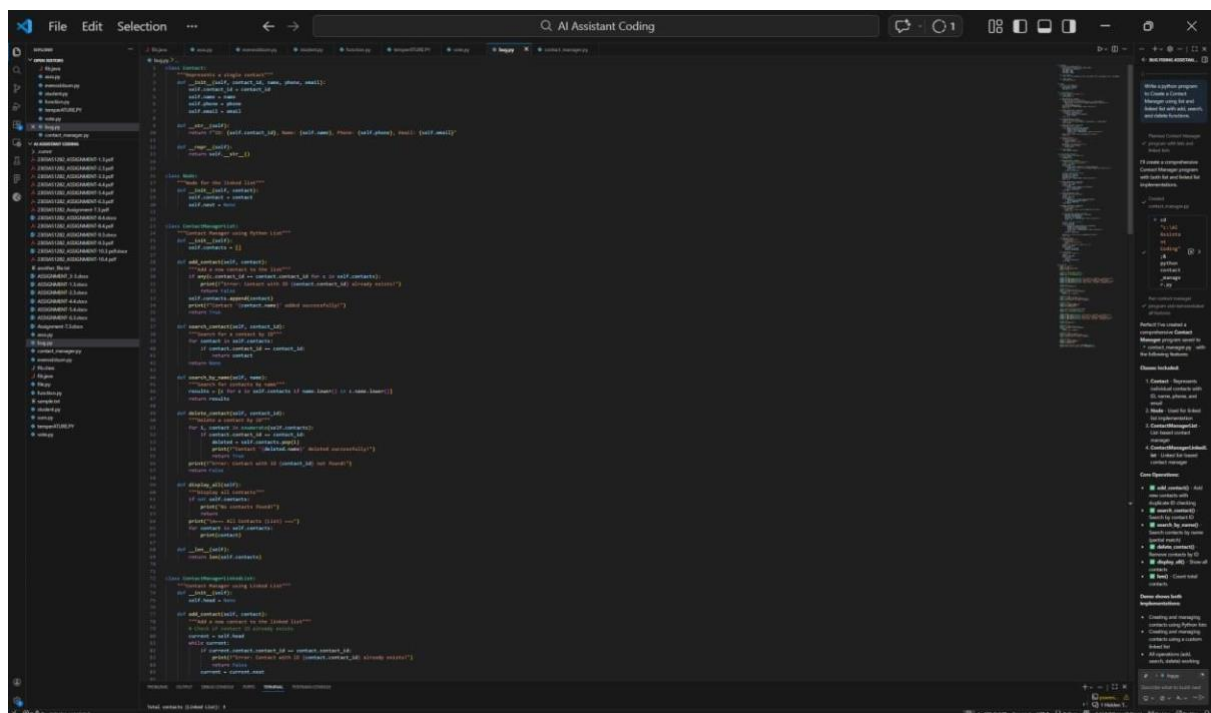
Scenario

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.

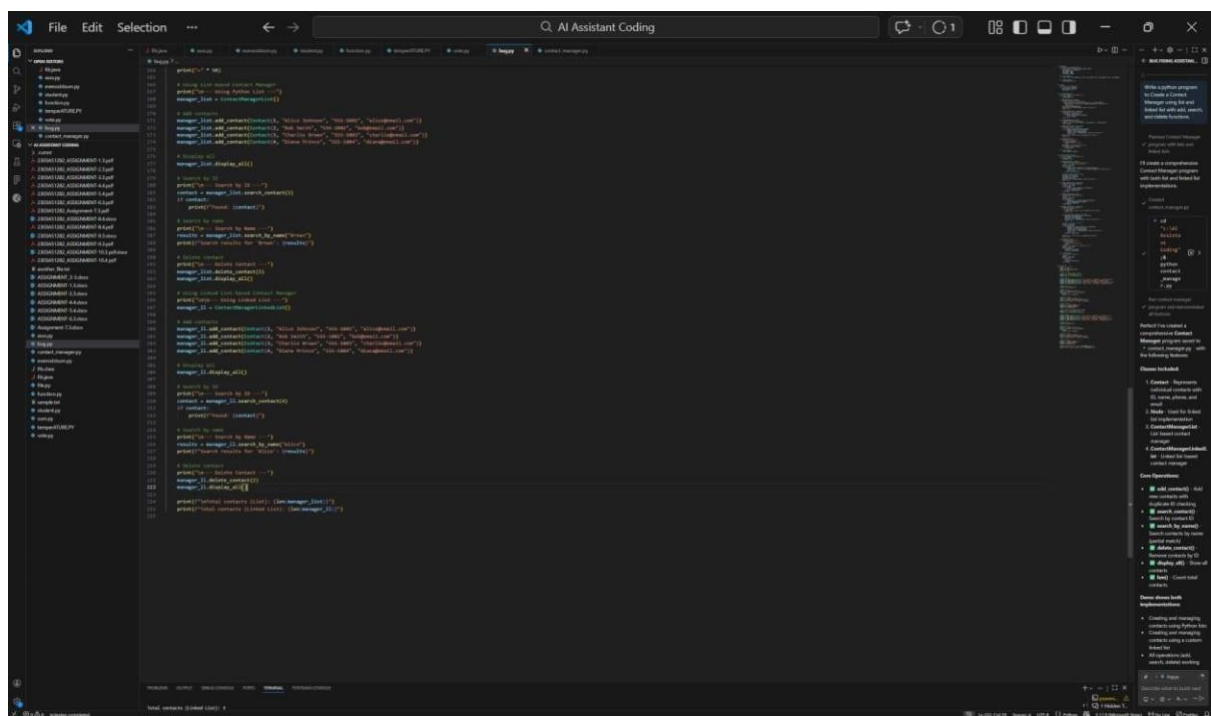
Prompt:

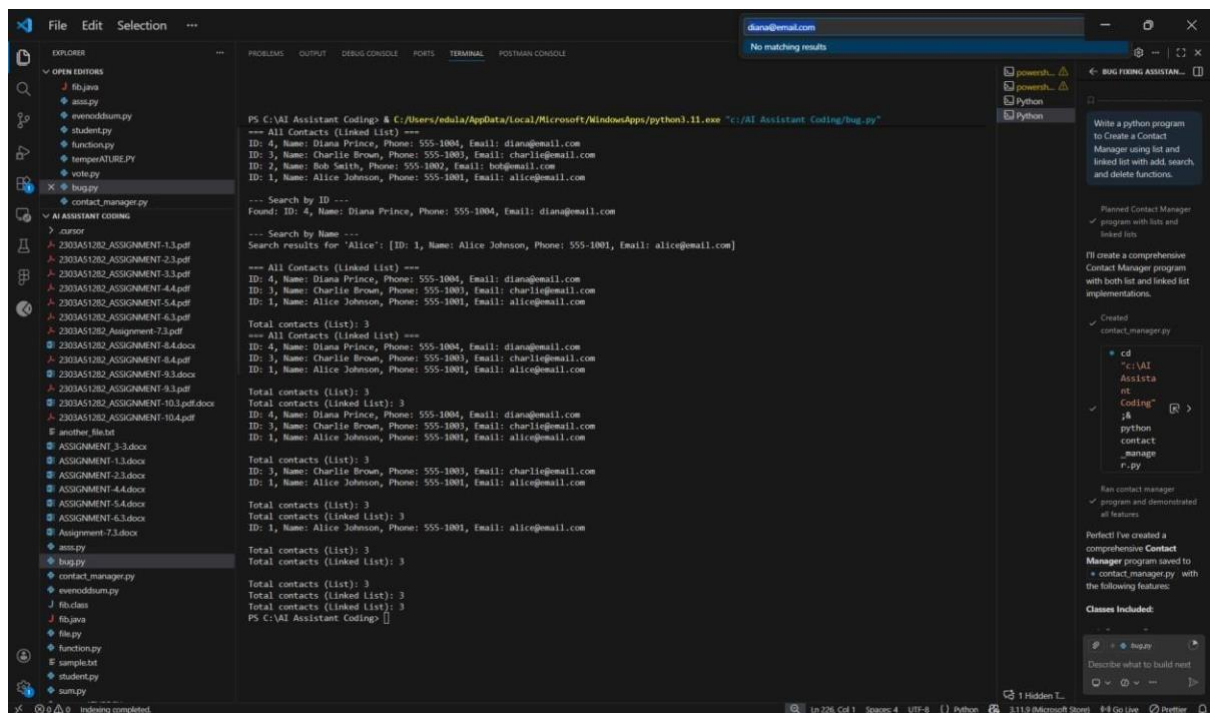
Write a python program to Create a Contact Manager using list and linked list with add, search, and delete functions.

Code:



```
1 class ContactManager:
2     def __init__(self):
3         self.head = None
4         self.tail = None
5         self.size = 0
6
7     def add_contact(self, name, phone, email):
8         """Add a new contact to the linked list"""
9         new_contact = Contact(name, phone, email)
10        if self.head is None:
11            self.head = new_contact
12            self.tail = new_contact
13        else:
14            self.tail.next = new_contact
15            self.tail = new_contact
16        self.size += 1
17
18    def search_contact(self, name):
19        """Search for a contact by name"""
20        current = self.head
21        while current:
22            if current.name == name:
23                return current
24            current = current.next
25        return None
26
27    def delete_contact(self, name):
28        """Delete a contact by name"""
29        if self.head is None:
30            return
31        if self.head.name == name:
32            self.head = self.head.next
33        else:
34            current = self.head
35            while current.next:
36                if current.next.name == name:
37                    current.next = current.next.next
38                current = current.next
39
40    def display_all_contacts(self):
41        """Display all contacts in the linked list"""
42        current = self.head
43        while current:
44            print(f"Name: {current.name}, Phone: {current.phone}, Email: {current.email}")
45            current = current.next
46
47    def __str__(self):
48        return f"Contact Manager with {self.size} contacts"
49
50    def __len__(self):
51        return self.size
52
53    def __iter__(self):
54        current = self.head
55        while current:
56            yield current
57            current = current.next
58
59    def __getitem__(self, index):
60        """Get contact by index"""
61        current = self.head
62        for _ in range(index):
63            current = current.next
64        return current
65
66    def __setitem__(self, index, value):
67        """Set contact by index"""
68        current = self.head
69        for _ in range(index):
70            current = current.next
71        current.name, current.phone, current.email = value
72
73    def __delitem__(self, index):
74        """Delete contact by index"""
75        current = self.head
76        for _ in range(index):
77            current = current.next
78        self.delete_contact(current.name)
```





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:


```

class PriorityQueue:
    def __init__(self):
        self.queue = []

    def enqueue(self, request):
        self.queue.append(request)

    def dequeue(self):
        if self.queue:
            return self.queue.pop(0)
        return None

    def display(self):
        print("Queue size: ", len(self.queue))

# ===== PRIORITY QUEUE =====
if __name__ == '__main__':
    p_queue = PriorityQueue()

    # Add requests
    p_queue.enqueue("Alice", "Python Programming")
    p_queue.enqueue("Bob", "Data Science")
    p_queue.enqueue("Charlie", "Web Development")
    p_queue.enqueue("David", "Machine Learning")

    # Process requests
    p_queue.dequeue()
    p_queue.dequeue()
    p_queue.dequeue()
    p_queue.dequeue()

    # Display queue
    p_queue.display()

    # Add requests (Faculty and Student)
    p_queue.enqueue("Prof. Smith", "Database Systems")
    p_queue.enqueue("Prof. Jones", "AI Research")
    p_queue.enqueue("Alice", "Python Programming")
    p_queue.enqueue("Bob", "Data Science")
    p_queue.enqueue("Charlie", "Web Development")
    p_queue.enqueue("David", "Machine Learning")

    # Process requests
    p_queue.dequeue()
    p_queue.dequeue()
    p_queue.dequeue()
    p_queue.dequeue()
    p_queue.dequeue()
    p_queue.dequeue()

    # Display queue
    p_queue.display()

```

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

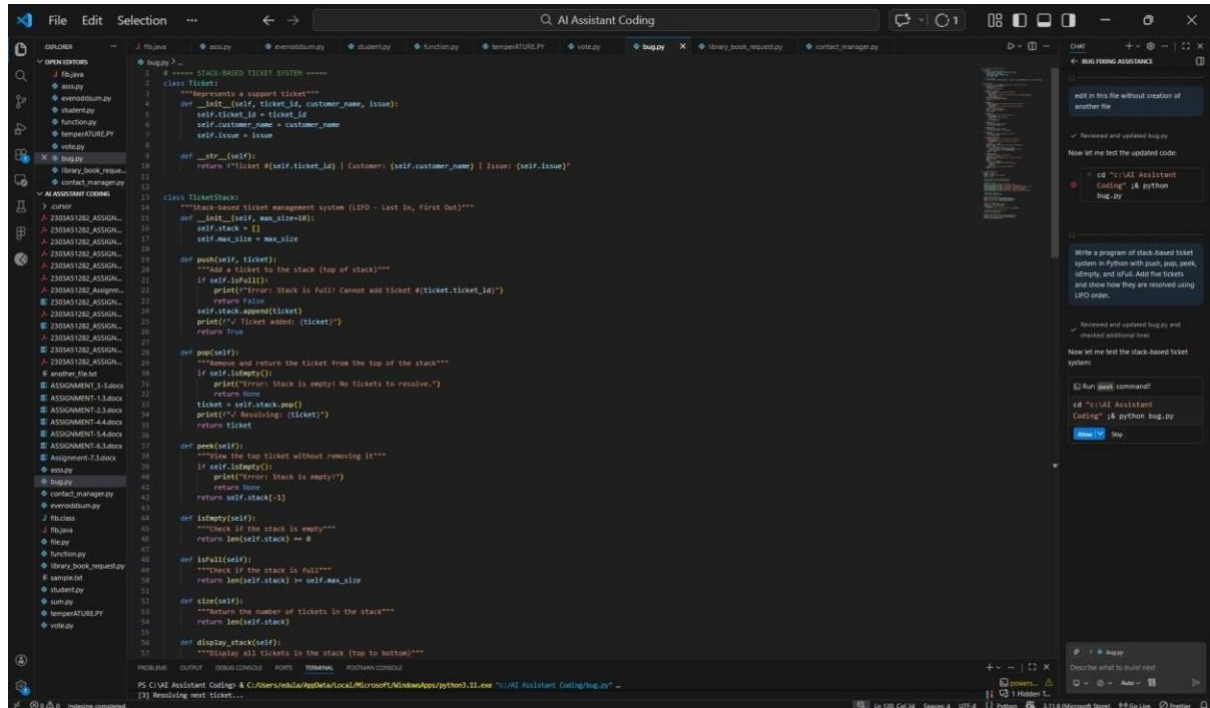
SR University's IT Help Desk receives technical support tickets from students and staff.

While tickets are received sequentially, issue escalation follows a Last-In, First-Out (LIFO) approach.

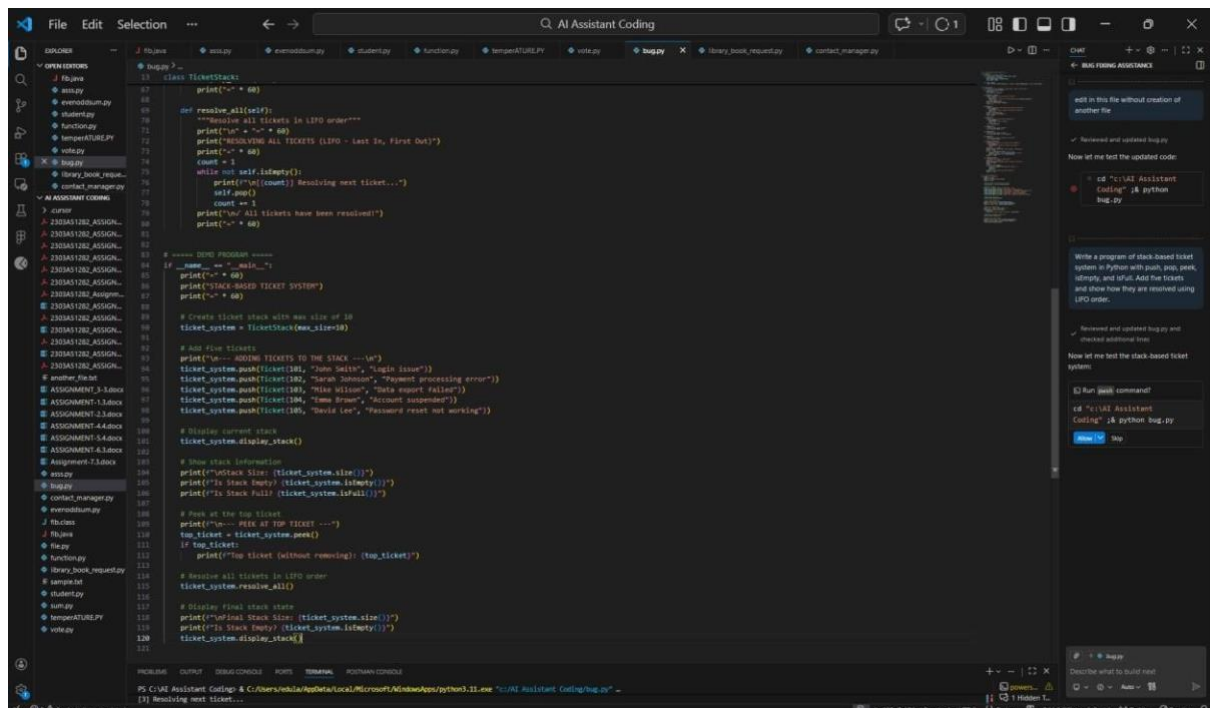
Prompt:

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:

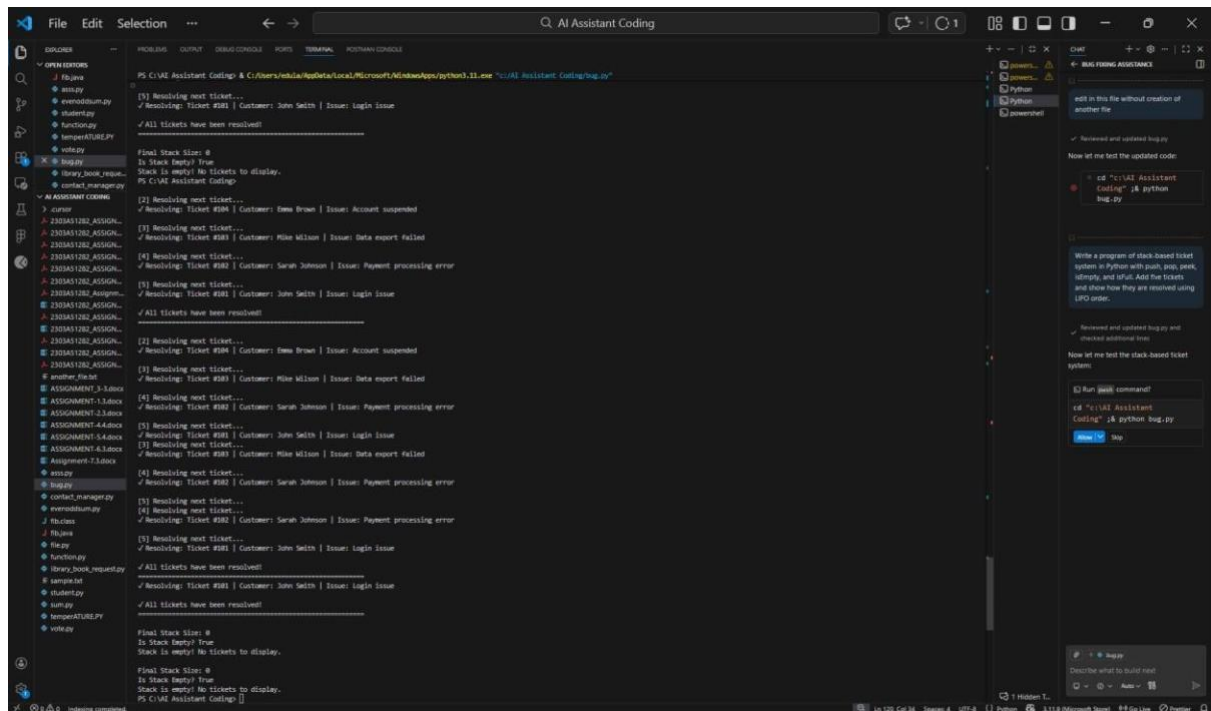


```
1 # ===== STACK-BASED TICKET SYSTEM =====
2 class Ticket:
3     """Represents a support ticket"""
4     def __init__(self, ticket_id, customer_name, issue):
5         self.ticket_id = ticket_id
6         self.customer_name = customer_name
7         self.issue = issue
8
9     def __str__(self):
10         return f"Ticket #{self.ticket_id} | Customer: {self.customer_name} | Issue: {self.issue}"
11
12 class TicketStack:
13     """Stack-based ticket management system (LIFO - Last In, First Out)"""
14     def __init__(self, max_size=10):
15         self.stack = []
16         self.max_size = max_size
17
18     def push(self, ticket):
19         """Add a ticket to the stack (top of stack)"""
20         if self.is_full():
21             print(f"Error: Stack is full! Cannot add ticket #{ticket.ticket_id}")
22             return False
23         self.stack.append(ticket)
24         print(f"Ticket added: {ticket}")
25         return True
26
27     def pop(self):
28         """Remove and return the ticket from the top of the stack"""
29         if self.is_empty():
30             print(f"Error: Stack is empty! No tickets to resolve.")
31             return None
32         ticket = self.stack.pop()
33         print(f"Resolving: {ticket}")
34         return ticket
35
36     def peek(self):
37         """View the top ticket without removing it"""
38         if self.is_empty():
39             print(f"Error: Stack is empty!")
40             return None
41         return self.stack[-1]
42
43     def is_empty(self):
44         """Check if the stack is empty"""
45         return len(self.stack) == 0
46
47     def is_full(self):
48         """Check if the stack is full"""
49         return len(self.stack) == self.max_size
50
51     def size(self):
52         """Return the number of tickets in the stack"""
53         return len(self.stack)
54
55     def display_stack(self):
56         """Display all tickets in the stack (top to bottom)"""
57         if not self.is_empty():
58             print("Current tickets in stack (top to bottom):")
59             for ticket in reversed(self.stack):
60                 print(ticket)
```



```
61 # ===== DEMO PROGRAM =====
62 if __name__ == "__main__":
63     print("=====")
64     print("STACK-BASED TICKET SYSTEM")
65     print("=====")
66
67     # Create ticket stack with max size of 10
68     ticket_system = TicketStack(max_size=10)
69
70     # Add five tickets
71     print("--- ADDING TICKETS TO THE STACK ---")
72     ticket_system.push(Ticket(182, "John Smith", "Login issue"))
73     ticket_system.push(Ticket(183, "Sarah Johnson", "Payment processing error"))
74     ticket_system.push(Ticket(184, "Mike Wilson", "Data export failed"))
75     ticket_system.push(Ticket(185, "Anna Brown", "Account suspension"))
76     ticket_system.push(Ticket(186, "David Lee", "Password reset not working"))
77
78     # Display current stack
79     ticket_system.display_stack()
80
81     # Show stack information
82     print(f"Stack Size: {ticket_system.size()}")
83     print(f"Is Stack Empty? {ticket_system.is_empty()}")
84     print(f"Is Stack Full? {ticket_system.is_full()}")
85
86     # Pop at the top ticket
87     print("--- PEAK AT TOP TICKET ---")
88     top_ticket = ticket_system.peek()
89     if top_ticket:
90         print(f"Top ticket (without removing): {top_ticket}")
91
92     # Resolve all tickets in LIFO order
93     ticket_system.resolve_all()
94
95     # Display final stack state
96     print(f"Final Stack Size: {ticket_system.size()}")
97     print(f"Is Stack Empty? {ticket_system.is_empty()}")
98     ticket_system.display_stack()
99
100 # =====
```

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:

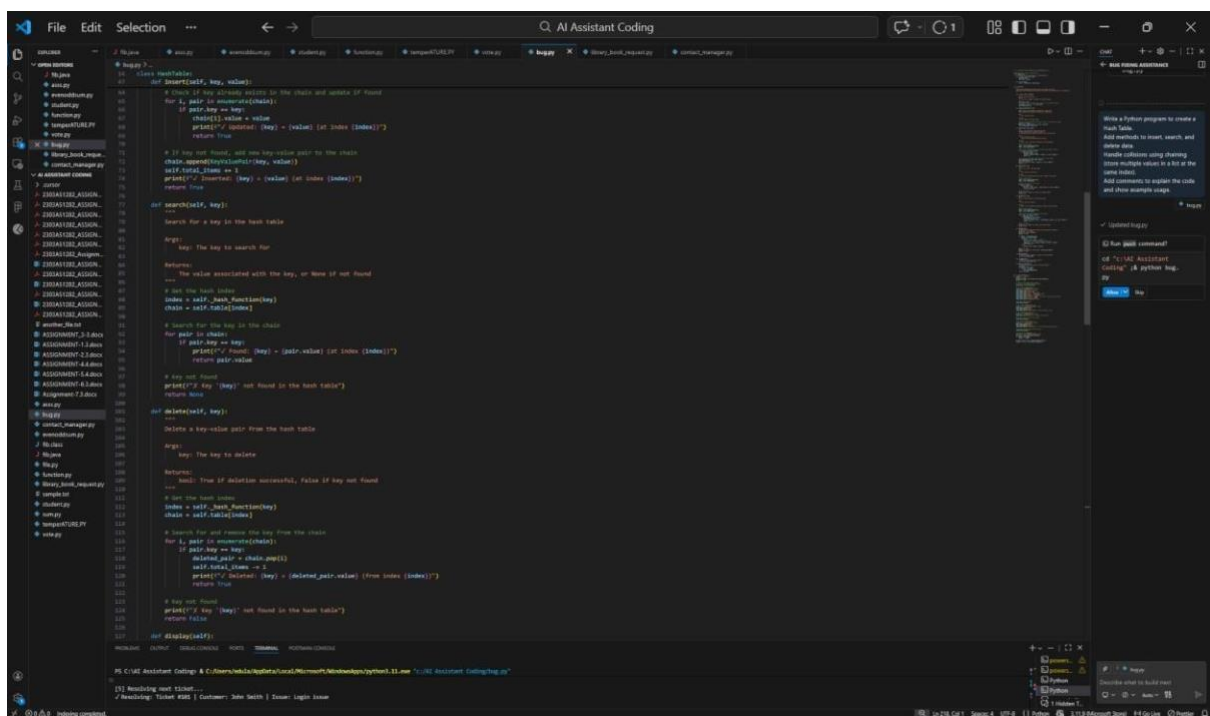
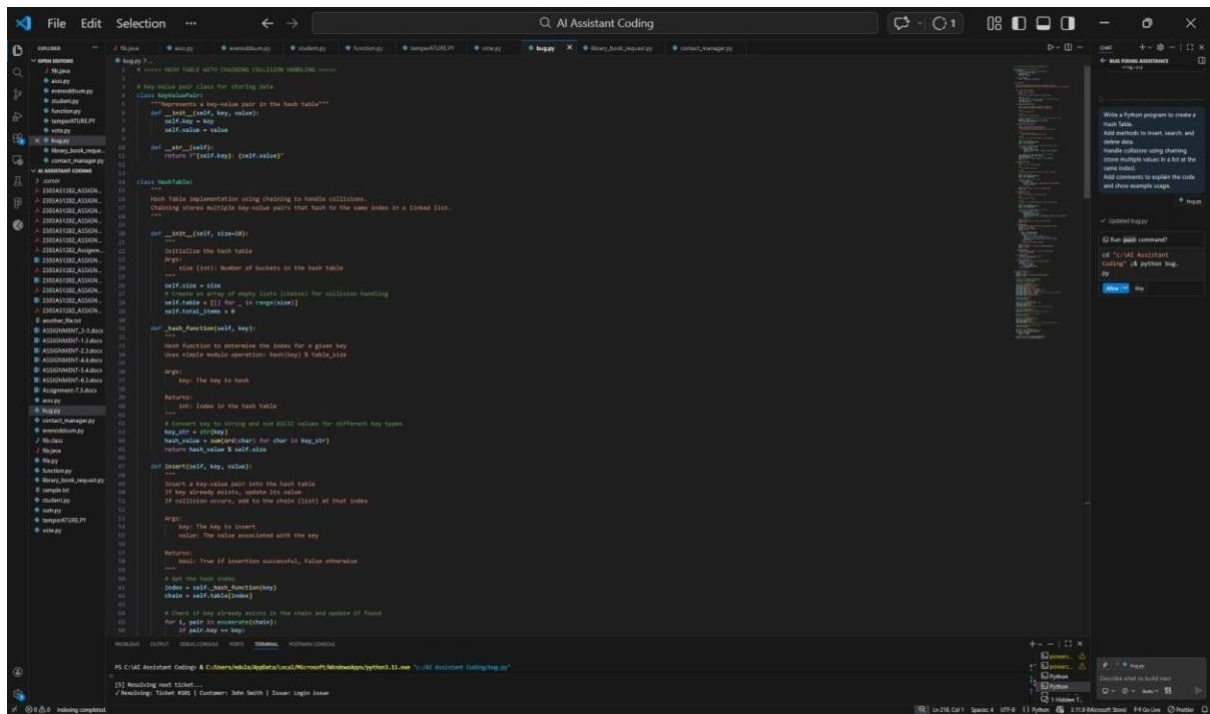
Write a Python program to create a Hash Table.

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:



Task 5:

Real-Time Application Challenge

Scenario

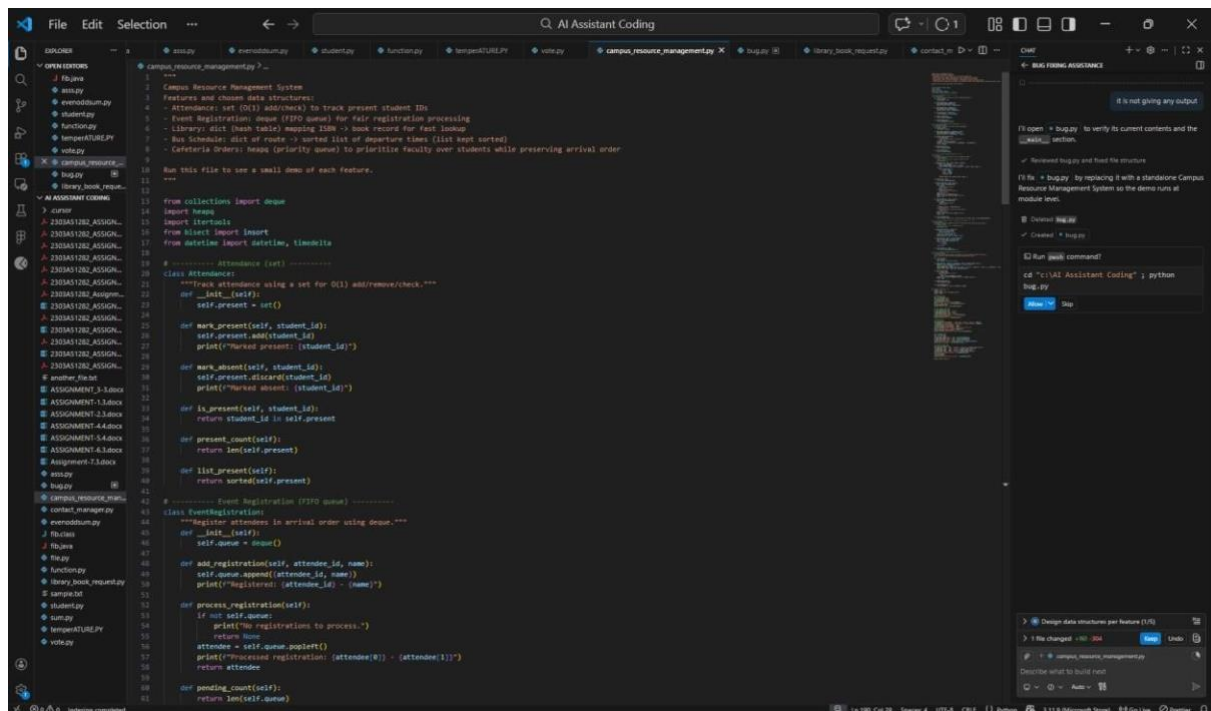
Design a Campus Resource Management System with the following features:

- 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:



```
1 # Campus Resource Management System
2 # Features and chosen data structures:
3 # - Attendance: set (O(1) add/check) to track present student IDs
4 # - Event Registration: queue (FIFO queue) for fair registration processing
5 # - Library: dict (hash table) mapping ISBN -> book record for fast lookup
6 # - Bus Schedule: dict of route -> sorted list of departure times (list kept sorted)
7 # - Cafeteria Orders: heapq (priority queue) to prioritize faculty over students while preserving arrival order
8
9 Run this file to see a small demo of each feature.
10
11 # ===== Attendance (set) =====
12
13 from collections import deque
14 import heapq
15 import itertools
16 from datetime import datetime, timedelta
17
18 class Attendance:
19     """Track attendance using a set for O(1) add/remove/check."""
20     def __init__(self):
21         self.present = set()
22
23     def mark_present(self, student_id):
24         self.present.add(student_id)
25         print(f"Marked present: {student_id}")
26
27     def mark_absent(self, student_id):
28         self.present.discard(student_id)
29         print(f"Marked absent: {student_id}")
30
31     def is_present(self, student_id):
32         return student_id in self.present
33
34     def present_count(self):
35         return len(self.present)
36
37     def list_present(self):
38         return sorted(self.present)
39
40 # ===== Event Registration (FIFO queue) =====
41
42 class EventRegistration:
43     """Register attendees in arrival order using deque."""
44     def __init__(self):
45         self.queue = deque()
46
47     def add_registration(self, attendee_id, name):
48         self.queue.append((attendee_id, name))
49         print(f"Registered: {attendee_id} - {name}")
50
51     def process_registration(self):
52         if not self.queue:
53             print("No registrations to process.")
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)
61
62 # ===== Library Book Request =====
63
64 class LibraryBookRequest:
65     """Manage book requests using a dictionary for ISBN lookup and a list for requests per ISBN."/>

```

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

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

# ----- Library (dict/hash table) -----
class Library:
    """Single 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['copies']) for isbn, info in self.catalog.items()]

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

```
# ----- Attendance -----
att = Attendance()
att.mark_present('0801')
att.mark_present('0802')
att.mark_present('0810')
print('Present list:', att.list_present())
print(f'0801 present', att.is_present('0801'))
att.mark_absent('0802')
print('Present count:', att.present_count())

# ----- Event registration -----
ev = EventRegistration()
ev.add_registration('0801', 'Alice')
ev.add_registration('0802', 'Bob')
ev.add_registration('0803', 'Charlie')
print('Pending registrations:', ev.list_pending())
ev.process_registration()
print('Pending count:', ev.pending_count())

# ----- Library -----
lib = Library()
lib.add_book('978-0130166307', 'Clean Code', 'Robert C. Martin', copies=2)
lib.borrow_book('978-0130166307', '0801')
lib.borrow_book('978-0130166307', '0802')
lib.borrow_book('978-0130166307', '0803')
lib.borrow_book('978-0130166307', '0804')
print('Available books:', lib.list_available())
lib.return_book('978-0130166307', '0801')
print('Available books after return:', lib.list_available())

# ----- Bus schedule -----
bs = BusSchedule()
now = datetime.now()
bs.add_route_time('Route A', now + timedelta(minutes=5))
bs.add_route_time('Route B', now + timedelta(minutes=10))
bs.add_route_time('Route C', now + timedelta(minutes=2))
print('Next Route A bus:', bs.next_bus('Route A', current_time=now))
print('Route A schedule:', bs.list_routes('Route A'))

# ----- Cafeteria orders -----
caf = CafeteriaOrders()
caf.add_order('0801', '0801', ['Coffee', 'Sandwich'], customer_type='Student')
caf.add_order('0802', '0801', ['Salad', 'Smoothie'], customer_type='Faculty')
caf.add_order('0803', '0802', ['Tea'], customer_type='Student')
caf.add_order('0804', '0802', ['Pastry'], 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('Cafeteria complete.')
```

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: 978-1491958296).
S001 borrowed Clean Code
S003 borrowed Clean Code
No copies available.
Available books: [('978-0135166307', 'Clean Code', 0), ('978-1491958296', 'Fluent Python', 1)]
S001 returned Clean Code
Available books after return: [('978-0135166307', 'Clean Code', 1), ('978-1491958296', 'Fluent Python', 1)]
Added bus time for Route A: 2026-02-18 18:42:24.367227
Added bus time for Route A: 2026-02-18 18:57:24.367227
Added bus time for Route B: 2026-02-18 18:39:24.367227
Next Route A bus: 2026-02-18 18:42:24.367227
Route A schedule: [datetime.datetime(2026, 2, 18, 18, 42, 24, 367227), datetime.datetime(2026, 2, 18, 18, 57, 24, 367227)]
Order added: 0001 (Student)
Order added: 0002 (Faculty)
Order added: 0003 (Student)
Order added: 0004 (Faculty)
Pending cafeteria orders: 4
Serving order: 0002 (Faculty)
Serving order: 0004 (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.