

Assignment 11.3 Ai Assisted Coding

Ht.no: 2303A510C3

Batch: 06

Task 1:

Smart Contact Manager (Arrays & Linked Lists)

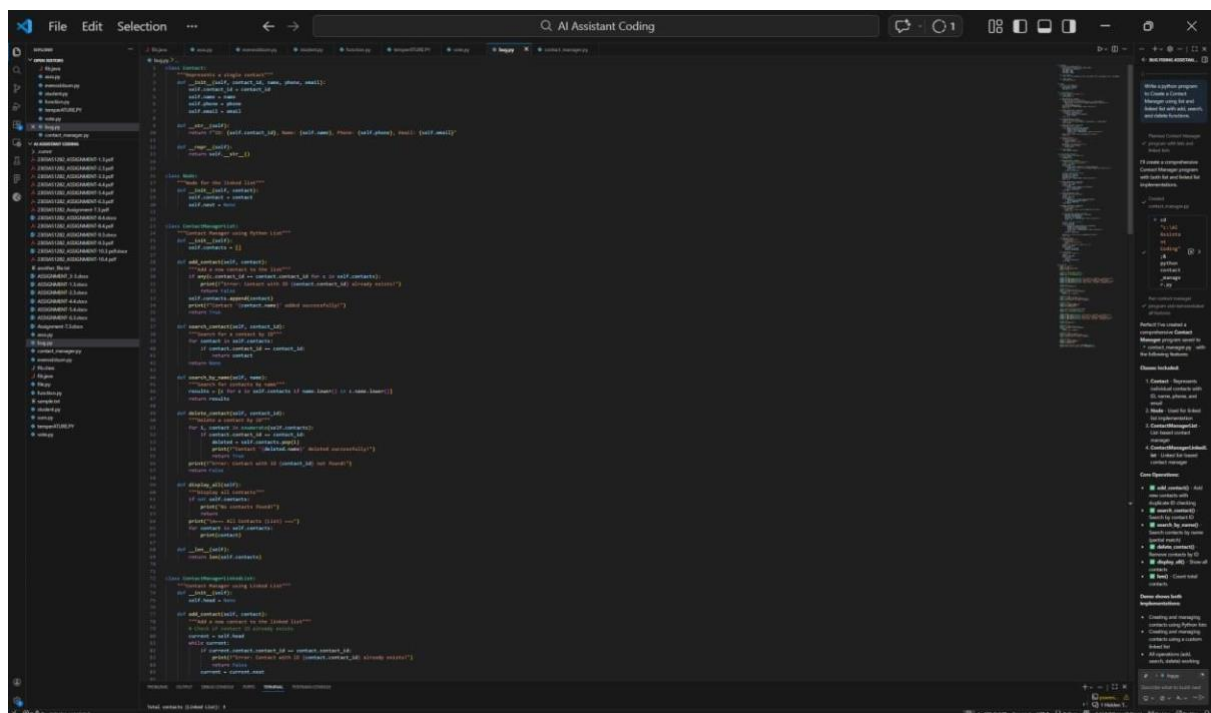
Scenario

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 Contact:
2     def __init__(self, name, phone, email):
3         self.name = name
4         self.phone = phone
5         self.email = email
6
7     def __str__(self):
8         return f"Contact: {self.name}, {self.phone}, {self.email}"
9
10 class ContactManager:
11     def __init__(self):
12         self._head = None
13
14     def add_contact(self, name, phone, email):
15         """Add a new contact to the linked list"""
16         new_contact = Contact(name, phone, email)
17         new_contact.next = self._head
18         self._head = new_contact
19
20     def search_contact(self, name):
21         """Search for a contact by name"""
22         current = self._head
23         while current:
24             if current.name == name:
25                 return current
26             current = current.next
27         return None
28
29     def delete_contact(self, name):
30         """Delete a contact by name"""
31         current = self._head
32         previous = None
33         while current:
34             if current.name == name:
35                 if previous:
36                     previous.next = current.next
37                 else:
38                     self._head = current.next
39                 return
40             previous = current
41             current = current.next
42         return None
43
44     def display_all_contacts(self):
45         """Display all contacts in the linked list"""
46         current = self._head
47         while current:
48             print(current)
49             current = current.next
50
51     def __str__(self):
52         return str(self._head)
```

```
#include <iostream>
#include <string>
#include <fstream>
#include <vector>
#include <map>
#include <algorithm>
using namespace std;

// Global variables
string filename;
vector<string> lines;
map<string, string> words;

// Function prototypes
void open_file();
void save_file();
void edit_file();
void display_menu();

// Main function
int main() {
    display_menu();
    while (true) {
        int choice;
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                open_file();
                break;
            case 2:
                save_file();
                break;
            case 3:
                edit_file();
                break;
            case 4:
                display_menu();
                break;
            default:
                cout << "Invalid choice. Please try again." << endl;
        }
    }
}
```

```
#include <iostream>
#include <string>
#include <fstream>
#include <vector>
#include <map>
#include <algorithm>
using namespace std;

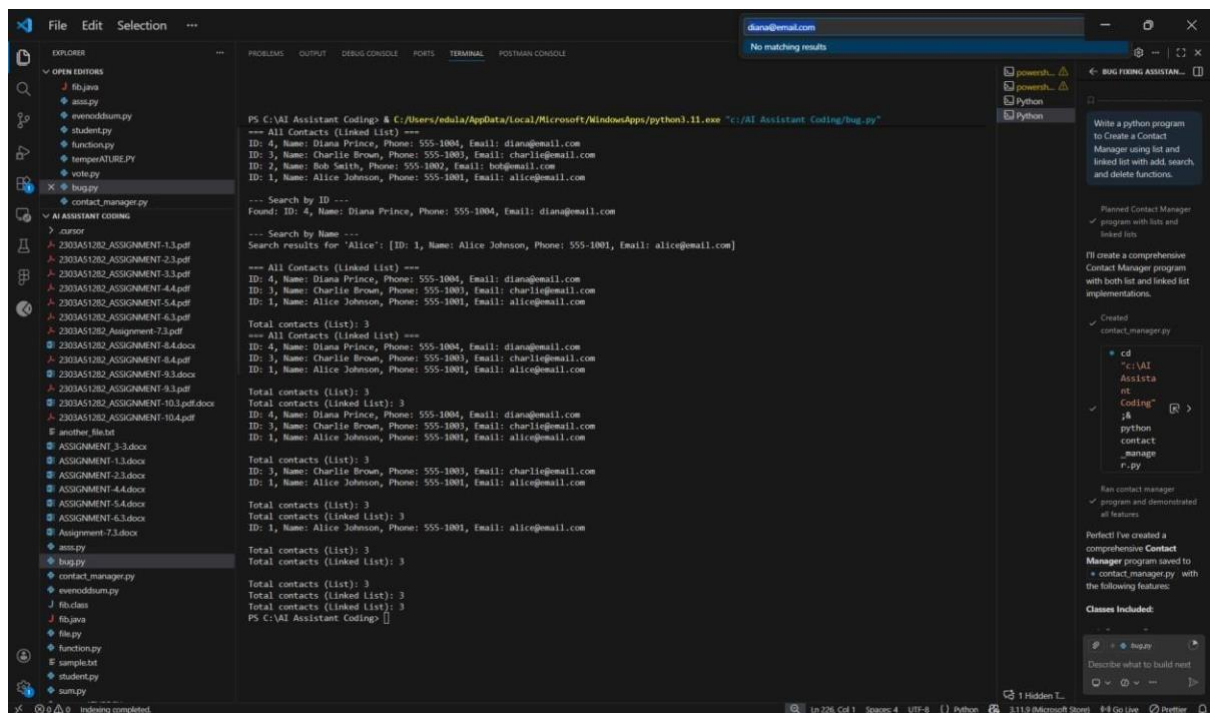
// Global variables
string filename;
vector<string> lines;
map<string, string> words;

// Function prototypes
void open_file();
void save_file();
void edit_file();
void display_menu();

// Main function
int main() {
    display_menu();
    while (true) {
        int choice;
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                open_file();
                break;
            case 2:
                save_file();
                break;
            case 3:
                edit_file();
                break;
            case 4:
                display_menu();
                break;
            default:
                cout << "Invalid choice. Please try again." << endl;
        }
    }
}
```

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:

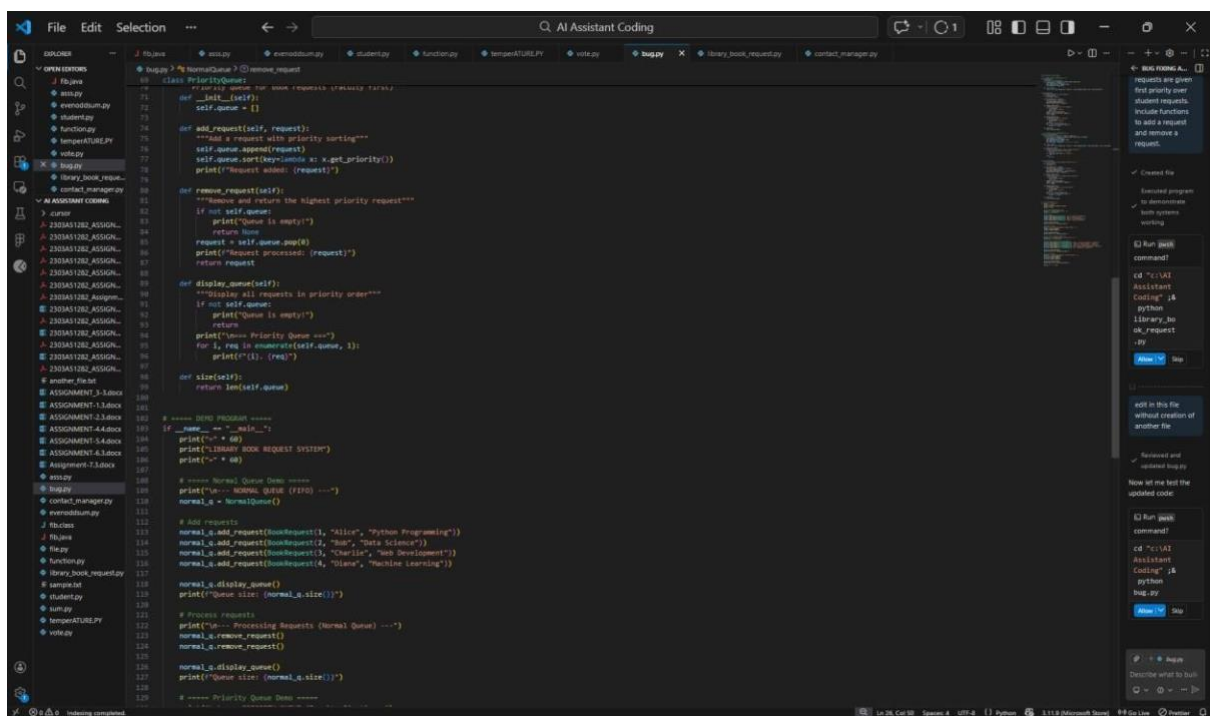
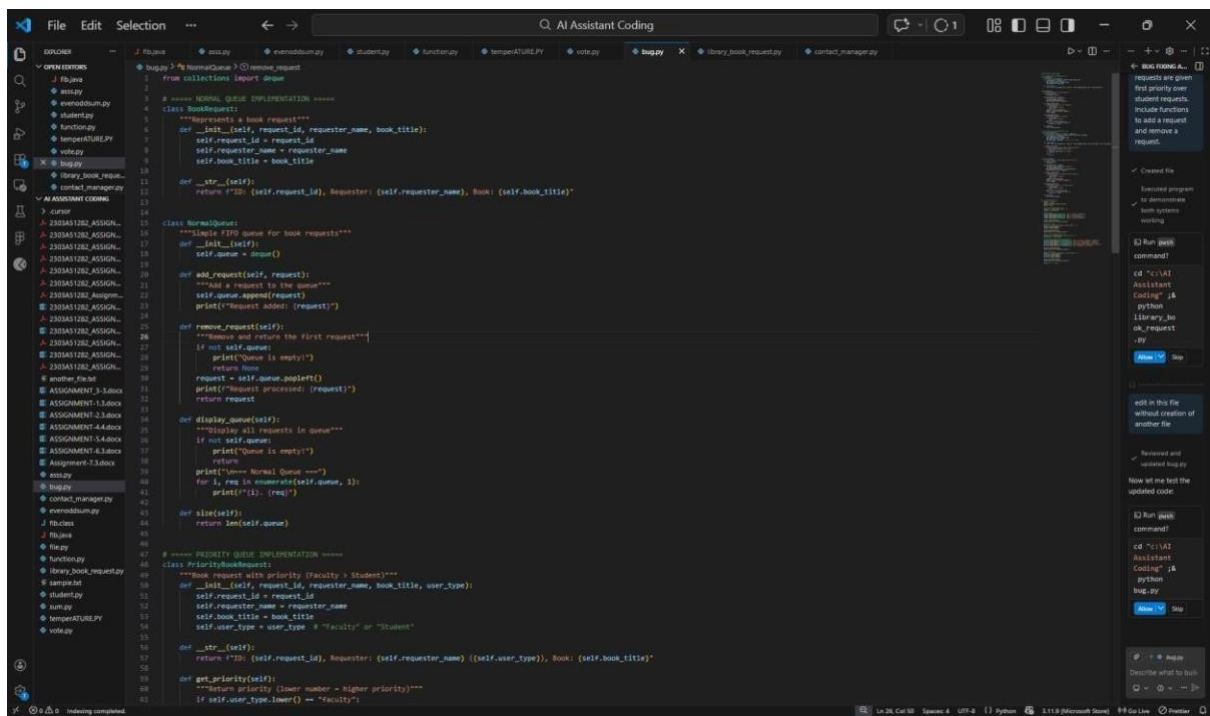
Library Book Search System (Queues & Priority Queues) Scenario

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:

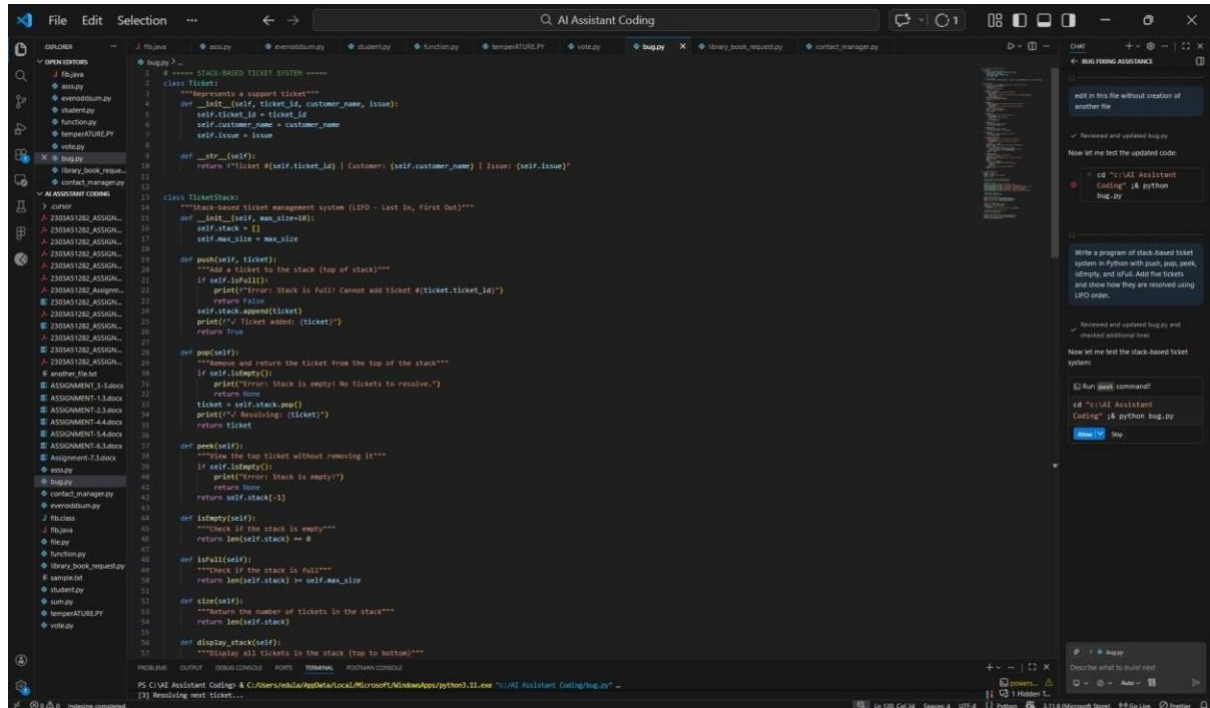


While tickets are received sequentially, issue escalation follows a Last-In, First-Out (LIFO) approach.

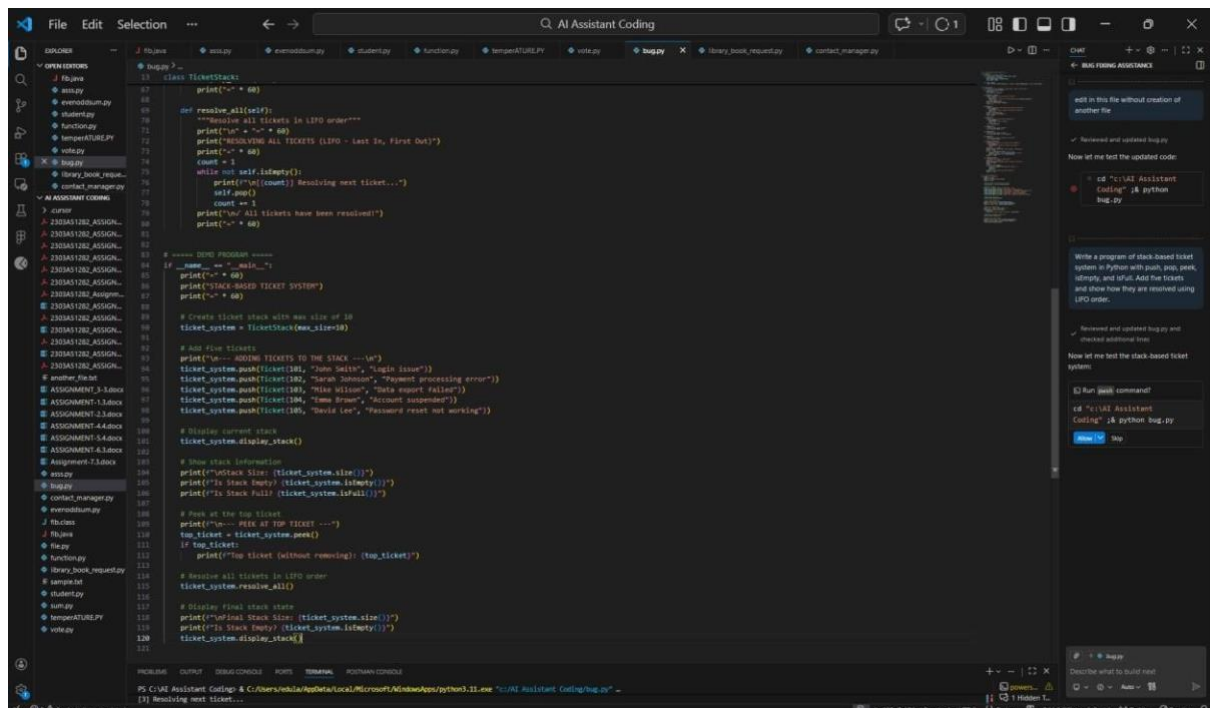
Prompt:

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:

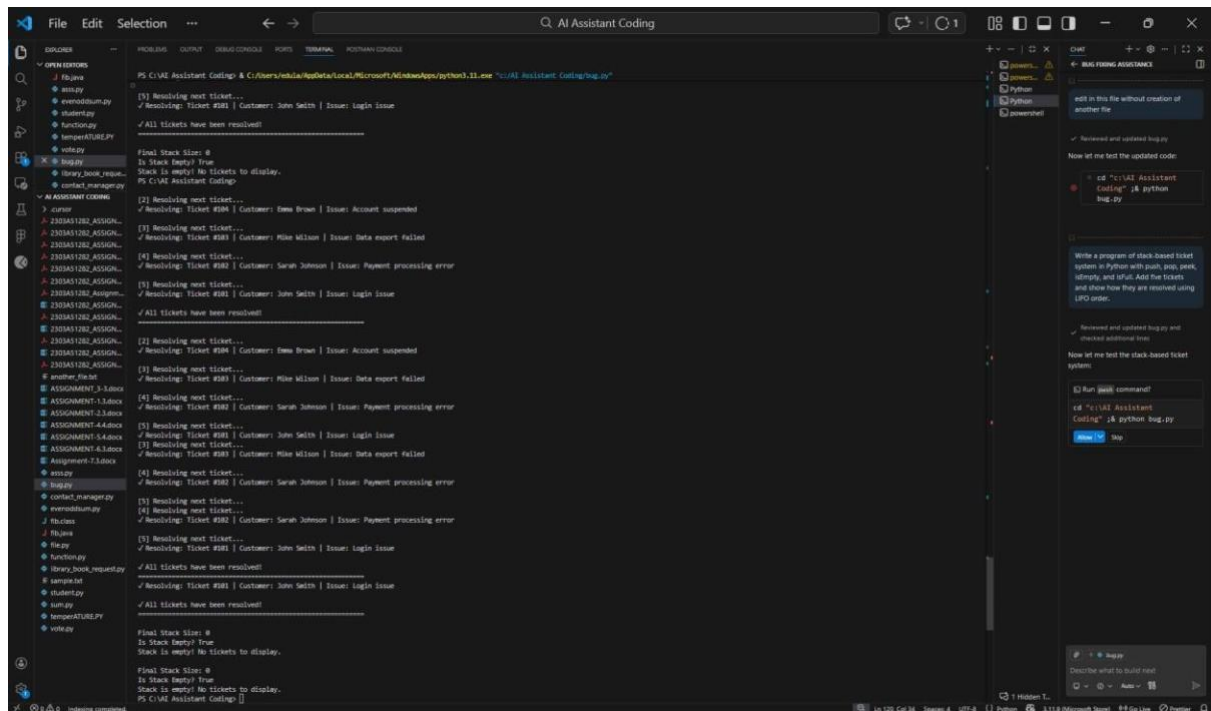


```
1 # ===== STACK-BASED TICKET SYSTEM =====
2 class Ticket:
3     """Represents a support ticket"""
4     def __init__(self, ticket_id, customer_name, issue):
5         self.ticket_id = ticket_id
6         self.customer_name = customer_name
7         self.issue = issue
8
9     def __str__(self):
10         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
62     def resolve_all(self):
63         """Resolve all tickets in LIFO order"""
64         print("Resolving ALL TICKETS (LIFO - Last In, First Out):")
65         count = 1
66         while not self.is_empty():
67             print(f"[{count}] Resolving next ticket...")
68             self.pop()
69             count += 1
70         print(f"Resolved: All tickets have been resolved!")
71
72 # ===== DEMO PROGRAM =====
73 if __name__ == "__main__":
74     print("=====")
75     print("STACK-BASED TICKET SYSTEM")
76     print("=====")
77
78     # Create ticket stack with max size of 10
79     ticket_system = TicketStack(max_size=10)
80
81     # Add five tickets
82     print("Adding tickets to the stack...")
83     ticket_system.push(Ticket(182, "John Smith", "Login issue"))
84     ticket_system.push(Ticket(183, "Sarah Johnson", "Payment processing error"))
85     ticket_system.push(Ticket(184, "Mike Wilson", "Data export failed"))
86     ticket_system.push(Ticket(185, "Anna Brown", "Account suspension"))
87     ticket_system.push(Ticket(186, "David Lee", "Password reset not working"))
88
89     # Display current stack
90     ticket_system.display_stack()
91
92     # Show stack information
93     print(f"Stack Size: {ticket_system.size()}")
94     print(f"Is Stack Empty? {ticket_system.is_empty()}")
95     print(f"Is Stack Full? {ticket_system.is_full()}")
96
97     # Peek at the top ticket
98     print("Peek at top ticket...")
99     top_ticket = ticket_system.peek()
100     if top_ticket:
101         print(f"Top ticket (without removing): {top_ticket}")
102
103     # Resolve all tickets in LIFO order
104     ticket_system.resolve_all()
105
106     # Display final stack state
107     print(f"Final Stack Size: {ticket_system.size()}")
108     print(f"Is Stack Empty? {ticket_system.is_empty()}")
109     ticket_system.display_stack()
110
111 # ===== TESTING =====
112 if __name__ == "__main__":
113     # Create ticket stack with max size of 10
114     ticket_system = TicketStack(max_size=10)
115
116     # Add five tickets
117     ticket_system.push(Ticket(182, "John Smith", "Login issue"))
118     ticket_system.push(Ticket(183, "Sarah Johnson", "Payment processing error"))
119     ticket_system.push(Ticket(184, "Mike Wilson", "Data export failed"))
120     ticket_system.push(Ticket(185, "Anna Brown", "Account suspension"))
121     ticket_system.push(Ticket(186, "David Lee", "Password reset not working"))
122
123     # Display current stack
124     ticket_system.display_stack()
125
126     # Show stack information
127     print(f"Stack Size: {ticket_system.size()}")
128     print(f"Is Stack Empty? {ticket_system.is_empty()}")
129     print(f"Is Stack Full? {ticket_system.is_full()}")
130
131     # Peek at the top ticket
132     print("Peek at top ticket...")
133     top_ticket = ticket_system.peek()
134     if top_ticket:
135         print(f"Top ticket (without removing): {top_ticket}")
136
137     # Resolve all tickets in LIFO order
138     ticket_system.resolve_all()
139
140     # Display final stack state
141     print(f"Final Stack Size: {ticket_system.size()}")
142     print(f"Is Stack Empty? {ticket_system.is_empty()}")
143     ticket_system.display_stack()
```

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:

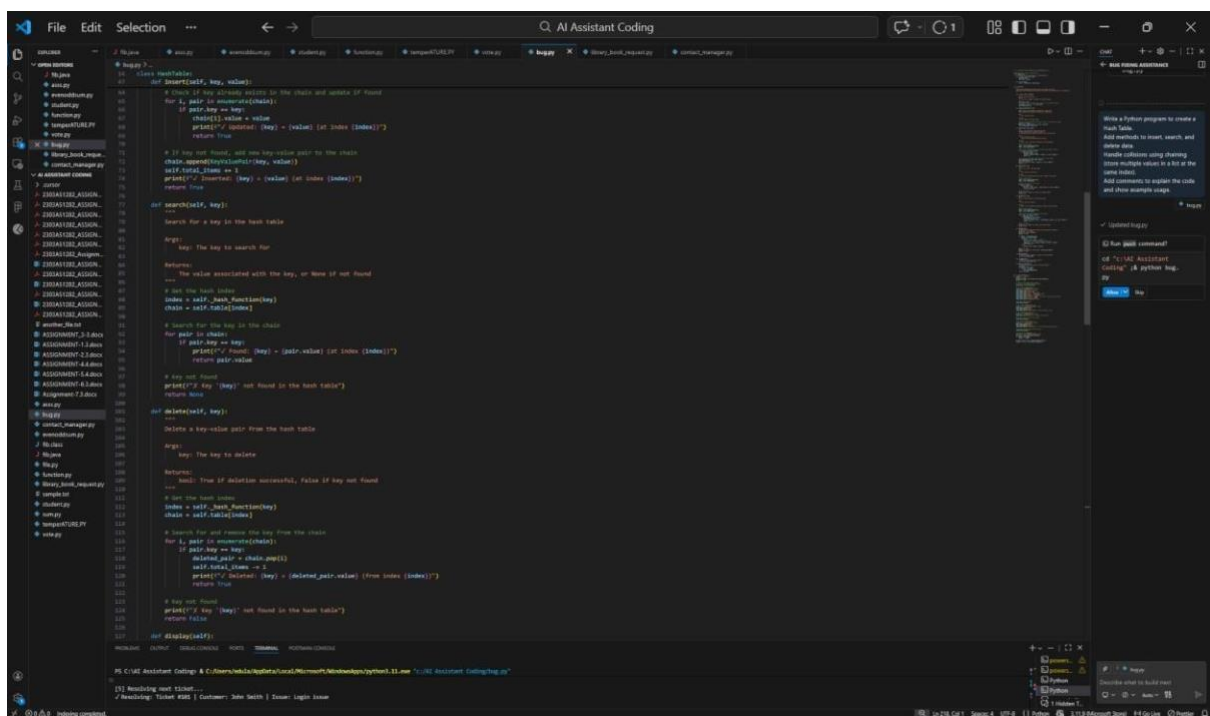
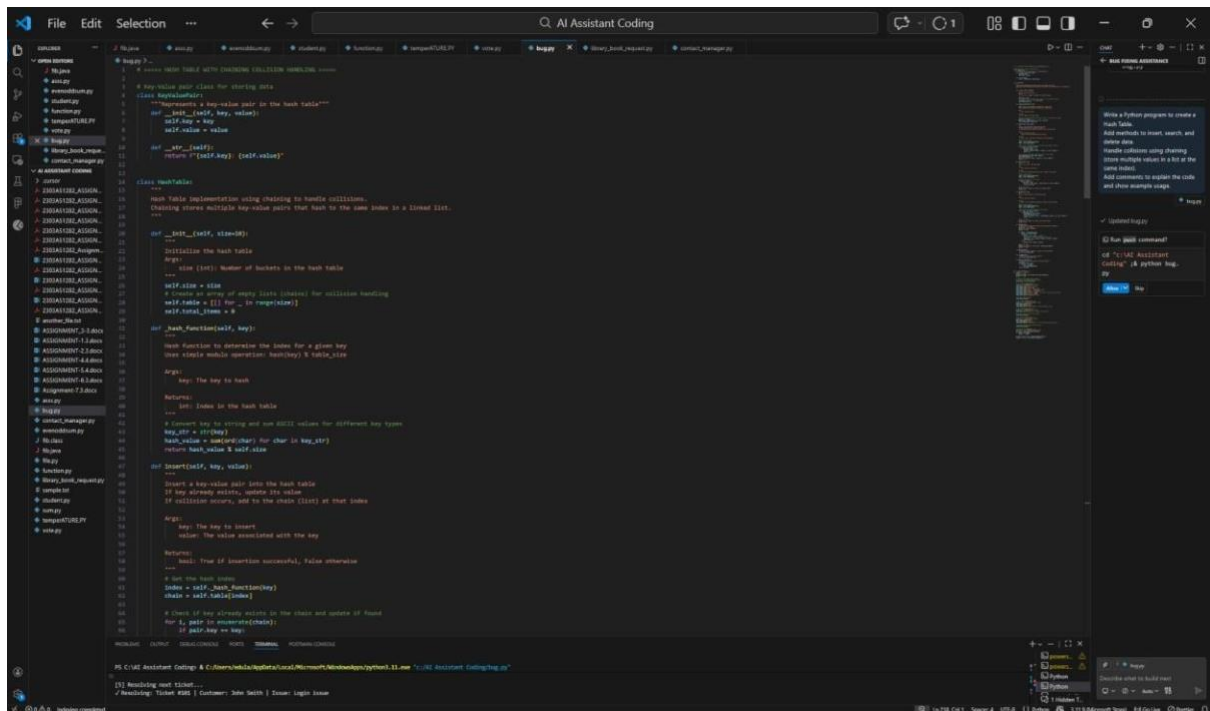
Write a Python program to create a Hash Table.

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 #!/usr/bin/env python3
2 """
3 Campus Resource Management System
4 Features and chosen data structures:
5 - Attendance: set (O(1) add/check) to track present student IDs
6 - Event Registration: queue (FIFO queue) for fair registration processing
7 - Library: dict (hash table) mapping ISBN -> book record for fast lookup
8 - Bus Schedule: dict of route -> sorted list of departure times (list kept sorted)
9 - Cafeteria Orders: heap (priority queue) to prioritize faculty over students while preserving arrival order
10 """
11
12 Run this file to see a small demo of each feature.
13
14 from collections import deque
15 import heapq
16 import itertools
17 from datetime import datetime, timedelta
18
19 # ===== Attendance (set) =====
20 class Attendance:
21     """Track attendance using a set for O(1) add/remove/check."""
22     def __init__(self):
23         self.present = set()
24
25     def mark_present(self, student_id):
26         self.present.add(student_id)
27         print(f"Marked present: {student_id}")
28
29     def mark_absent(self, student_id):
30         self.present.discard(student_id)
31         print(f"Marked absent: {student_id}")
32
33     def is_present(self, student_id):
34         return student_id in self.present
35
36     def present_count(self):
37         return len(self.present)
38
39     def list_present(self):
40         return sorted(self.present)
41
42 # ===== Event Registration (FIFO queue) =====
43 class EventRegistration:
44     """Register attendees in arrival order using deque."""
45     def __init__(self):
46         self.queue = deque()
47
48     def add_registration(self, attendee_id, name):
49         self.queue.append((attendee_id, name))
50         print(f"Registered: {attendee_id} - {name}")
51
52     def process_registration(self):
53         if not self.queue:
54             print("No registrations to process.")
55             return None
56         attendee = self.queue.popleft()
57         print(f"Processed registration: {attendee[0]} - {attendee[1]}")
58         return attendee
59
60     def pending_count(self):
61         return len(self.queue)
62
63 # ===== Library Book Request (dict) =====
64 class LibraryBookRequest:
65     """Map ISBNs to book records (dict of ISBN -> {title, author, status})."""
66     def __init__(self):
67         self.books = {}
68
69     def add_book(self, isbn, title, author):
70         self.books[isbn] = {"title": title, "author": author, "status": "available"}
71         print(f"Added book: {isbn} - {title} by {author}")
72
73     def get_book(self, isbn):
74         return self.books.get(isbn)
75
76     def request_book(self, isbn):
77         book = self.get_book(isbn)
78         if book and book["status"] == "available":
79             book["status"] = "borrowed"
80             print(f"Book {isbn} requested and marked borrowed.")
81         else:
82             print(f"Book {isbn} not found or already borrowed.")
83
84 # ===== Bus Schedule (dict of route -> sorted list) =====
85 class BusSchedule:
86     """Map routes to sorted lists of departure times (list kept sorted)."""
87     def __init__(self):
88         self.schedule = {}
89
90     def add_route(self, route, departure_time):
91         if route not in self.schedule:
92             self.schedule[route] = []
93         self.schedule[route].append(departure_time)
94         self.schedule[route].sort()
95         print(f"Added route {route} with departure time {departure_time}")
96
97     def get_departure_times(self, route):
98         return self.schedule.get(route, [])
99
100 # ===== Cafeteria Order Queue (heap) =====
101 class CafeteriaOrderQueue:
102     """Priority queue for cafeteria orders using a heap. Priority: Faculty > Student."""
103     def __init__(self):
104         self.orders = []
105
106     def add_order(self, priority, item):
107         # Priority: Faculty (1) > Student (2)
108         heapq.heappush(self.orders, (priority, item))
109         print(f"Added order: {priority} - {item}")
110
111     def get_order(self):
112         if self.orders:
113             priority, item = heapq.heappop(self.orders)
114             print(f"Processed order: {priority} - {item}")
115             return item
116         return None
117
118 # ===== Main Demo =====
119 def main():
120     attendance = Attendance()
121     event_reg = EventRegistration()
122     library = LibraryBookRequest()
123     bus = BusSchedule()
124     cafeteria = CafeteriaOrderQueue()
125
126     # Attendance Demo
127     attendance.mark_present(230AS1282)
128     attendance.mark_present(230AS1282)
129     attendance.mark_absent(230AS1282)
130     print(f"Present count: {attendance.present_count()}")
131     print(f"List present: {attendance.list_present()}")
132
133     # Event Registration Demo
134     event_reg.add_registration(230AS1282, "Alice")
135     event_reg.add_registration(230AS1282, "Bob")
136     event_reg.add_registration(230AS1282, "Charlie")
137     while event_reg.pending_count() > 0:
138         event_reg.process_registration()
139
140     # Library Demo
141     library.add_book("9780130358195", "Python Crash Course", "Eric Matthes")
142     library.request_book("9780130358195")
143     library.get_book("9780130358195")
144
145     # Bus Schedule Demo
146     bus.add_route("Route 1", datetime(2024, 1, 1, 8, 0))
147     bus.add_route("Route 1", datetime(2024, 1, 1, 8, 15))
148     bus.add_route("Route 1", datetime(2024, 1, 1, 8, 30))
149     print(f"Route 1 departure times: {bus.get_departure_times('Route 1')}")
150
151     # Cafeteria Order Demo
152     cafeteria.add_order(1, "Faculty: Coffee")
153     cafeteria.add_order(2, "Student: Sandwich")
154     cafeteria.add_order(1, "Faculty: Juice")
155     while True:
156         order = cafeteria.get_order()
157         if order is None:
158             break
159
160 if __name__ == "__main__":
161     main()
```

```
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):
            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) -----
```

```
def main():
    # ----- Attendance -----
    att = Attendance()
    att.mark_present("0801")
    att.mark_present("0802")
    att.mark_present("0803")
    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")
    print(f"Available books:", lib.list_available())
    lib.return_book("978-0130166307", "0801")
    print("Available books after return:", lib.list_available())

    # ----- Bus schedule -----
    bs = BusSchedule()
    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", "0802", ["Salad", "Fruit"], customer_type="Faculty")
    caf.add_order("0803", "0803", ["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.