

AI ASSISTANT CODING

ASSIGNMENT - 11.3

Name: Aman Sarkar

Ht.no: 2303A51273

Batch: 05

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

Scenario

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

Prompt:

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

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

Code:

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface with the following details:

- File Bar:** File, Edit, Selection, Back, Forward, Search bar (AI Assistant Coding), Home, Help.
- Sidebar:** Shows file navigation, including a tree view of files like `__init__.py`, `main.py`, `menu.py`, `contact.py`, `contactDB.py`, and `log.py`.
- Code Editor:** Displays Python code for a contact management system. The code includes functions for adding, searching, updating, and deleting contacts from a database. It uses classes like `Contact` and `ContactManager`. A specific function `add_contact(self, contact)` is highlighted.
- AI Assistant Coding Panel:** Located on the right side, it provides suggestions for the current code:
 - Write a python program to manage contacts
 - Manage contacts list and search
 - FI make a comprehensive Contact Manager program
 - Overall
- Output Panel:** Shows logs related to the AI Assistant Coding feature.
- Bottom Status Bar:** Includes tabs for Python 3, Python 2, and Python 3 environments, along with other status indicators.

Output:

The screenshot shows the Microsoft Visual Studio Code interface with the following details:

- File Explorer:** On the left, it lists various files including `fb.java`, `evenodisum.py`, `student.py`, `function.py`, `temperature.py`, `vole.py`, `bug.py`, `contact_manager.py`, and several assignment files like `ASSIGNMENT-13.pdf` through `ASSIGNMENT-10.pdf`.
- Terminal:** The main terminal window displays Python code for a contact manager. It includes functions for searching by ID, name, or contact list, and demonstrates how to add contacts to a linked list.
- Status Bar:** At the bottom, it shows the path `C:\AI Assistant Coding & C:\Users\diana\AppData\Local\Microsoft\WindowsApps\python3.11.exe`, the command `"c:\AI Assistant Coding\bug.py"`, and the status `Indexing completed.`
- Search Bar:** The search bar at the top right contains the email address `diana@email.com` and shows the result `No matching results`.
- Code Snippets:** A sidebar on the right titled "BUG FIXING ASSISTANT..." provides snippets for writing a Python Contact Manager program.
- Bottom Status:** The bottom right corner shows the Microsoft Store logo and the text "1 Hidden...".

Explanation:

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

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

Scenario

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

Prompt:

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

Write a Python program for a library book request system.

First, make a normal queue where requests are handled in the order they come.

Then, make another version where faculty requests are given first priority over student requests.

Include functions to add a request and remove a request.

Code:

```
# ----- NORMAL QUEUE IMPLEMENTATION -----
class BookRequest:
    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:
    """Single FIFO queue for book requests"""
    def __init__(self):
        self.queue = []

    def add_request(self, request):
        """Add a request to the queue"""
        self.queue.append(request)
        print("Request added: " + str(request))

    def remove_request(self):
        """Remove and return the first request"""
        if not self.queue:
            print("Queue is empty")
            return None
        else:
            request = self.queue.pop(0)
            print("Request processed: " + str(request))
            return request

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


# ----- 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":
            return 1
        else:
            return 2

# ----- DEMO PROGRAM -----
if __name__ == "__main__":
    print("+" * 40)
    print("LIBRARY BOOK REQUEST SYSTEM")
    print("+" * 40)

    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("Queue size: " + str(normal_q.size()))

    # Process requests
    print("+" * 40)
    print("Processing Requests (Normal Queue) ...")
    normal_q.remove_request()
    normal_q.remove_request()
    normal_q.remove_request()

    normal_q.display_queue()
    print("Queue size: " + str(normal_q.size()))

    # ----- Priority Queue Demo -----
    print("+" * 40)
    print("Priority Queue Demo")

    priority_q = PriorityBookRequest("P1", "Faculty", "Python Programming", "Faculty")
    priority_q = PriorityBookRequest("P2", "Student", "Data Science", "Student")
    priority_q = PriorityBookRequest("P3", "Faculty", "Web Development", "Faculty")
    priority_q = PriorityBookRequest("P4", "Student", "Machine Learning", "Student")
```

```
# ----- NORMAL QUEUE (FIFO) -----
class NormalQueue:
    def __init__(self):
        self.queue = []

    def add_request(self, request):
        """Add a request to the queue"""
        self.queue.append(request)
        print("Request added: " + str(request))

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

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


# ----- PRIORITY QUEUE -----
class PriorityQueue:
    """Book request with priority (Faculty > Student)"""
    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("Request added: " + str(request))

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

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


# ----- DEMO PROGRAM -----
if __name__ == "__main__":
    print("+" * 40)
    print("LIBRARY BOOK REQUEST SYSTEM")
    print("+" * 40)

    # ----- Normal Queue Demo -----
    print("+" * 40)
    print("Normal Queue Demo")
    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("Queue size: " + str(normal_q.size()))

    # Process requests
    print("+" * 40)
    print("Processing Requests (Normal Queue) ...")
    normal_q.remove_request()
    normal_q.remove_request()
    normal_q.remove_request()

    normal_q.display_queue()
    print("Queue size: " + str(normal_q.size()))

    # ----- Priority Queue Demo -----
    print("+" * 40)
    print("Priority Queue Demo")

    priority_q = PriorityBookRequest("P1", "Faculty", "Python Programming", "Faculty")
    priority_q = PriorityBookRequest("P2", "Student", "Data Science", "Student")
    priority_q = PriorityBookRequest("P3", "Faculty", "Web Development", "Faculty")
    priority_q = PriorityBookRequest("P4", "Student", "Machine Learning", "Student")
```

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

=====
```

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.

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:

File Edit Selection ...

AI Assistant Coding

DISCOVERY OPEN EDITORS

J fb.java assipy evenoddsum.py student.py function.py temperature.py vote.py buggy library_book_request.py contact_manager.py

chat bug fixing assistance

edit in this file without creation of another file

✓ Reviewed and updated buggy

Now let me test the updated code!

cd "c:\AI Assistant Coding" & python buggy

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 buggy and checked additional files

Now let me test the stack-based ticket system!

Run [will command] cd "c:\AI Assistant Coding" & python buggy

Allow Skip

```

bugpy > ...
1 # ----- STACK-BASED TICKET SYSTEM -----
2 class TicketStack:
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 LIFOStack:
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.isFull():
21             print("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.isEmpty():
30             print("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.isEmpty():
39             print("Error: Stack is empty!")
40         return self.stack[-1]
41
42     def isEmpty(self):
43         """Check if the stack is empty"""
44         return len(self.stack) == 0
45
46     def isFull(self):
47         """Check if the stack is full"""
48         return len(self.stack) >= self.max_size
49
50     def size(self):
51         """Return the number of tickets in the stack"""
52         return len(self.stack)
53
54     def display_stack(self):
55         """Display all tickets in the stack (top to bottom)"""
56
57 PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL POSTMAN CONSOLE
PS C:\AI Assistant Coding & C:\Users\edula\AppData\Local\Microsoft\Al\EndUser\python3.11.exe "c:\AI Assistant Coding\bug.py"
[3] Resolving next ticket...

```

File Edit Selection ...

AI Assistant Coding

DISCOVERY OPEN EDITORS

J fb.java assipy evenoddsum.py student.py function.py temperature.py vote.py buggy library_book_request.py contact_manager.py

chat bug fixing assistance

edit in this file without creation of another file

✓ Reviewed and updated buggy

Now let me test the updated code!

cd "c:\AI Assistant Coding" & python buggy

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 buggy and checked additional files

Now let me test the stack-based ticket system!

Run [will command] cd "c:\AI Assistant Coding" & python buggy

Allow Skip

```

bugpy > ...
15 class TicketStack:
16     print("+" * 60)
17
18     def resolve_all(self):
19         """Resolve all tickets in LIFO order"""
20         print("\n" + "-" * 60)
21         print("RESOLVING ALL TICKETS (LIFO - Last In, First Out)")
22         print("\n" + "-" * 60)
23         count = 0
24         while not self.isEmpty():
25             print(f"\n{count}) Resolving next ticket...")
26             self.pop()
27             count += 1
28         print("\n" + "-" * 60)
29         print("+" * 60)
30
31     # ----- DEBO PROGRAM -----
32     if __name__ == "__main__":
33         print("-" * 60)
34         print("+" * 60)
35         print("STACK-BASED TICKET SYSTEM")
36         print("-" * 60)
37
38         # Create ticket stack with max size of 10
39         ticket_system = TicketStack(max_size=10)
40
41         # Add five tickets
42         print("\n" + "-" * 60)
43         print("... ADDING TICKETS TO THE STACK ---")
44         ticket_system.push(Ticket(101, "John Smith", "Login issue"))
45         ticket_system.push(Ticket(102, "Sarah Johnson", "Payment processing error"))
46         ticket_system.push(Ticket(103, "Michael Brown", "Ticket support failed"))
47         ticket_system.push(Ticket(104, "Emma Green", "Account suspended"))
48         ticket_system.push(Ticket(105, "David Lee", "Password reset not working"))
49
50         # Display current stack
51         ticket_system.display_stack()
52
53         # Show stack information
54         print("\n" + "-" * 60)
55         print(f"Stack Size: {ticket_system.size()}")
56         print(f"Is Stack Empty? {ticket_system.isEmpty()}")
57         print(f"Is Stack Full? {ticket_system.isFull()}")
58
59         # Peek at the top ticket
60         print("\n" + "-" * 60)
61         print("... PEAK AT TOP TICKET ---")
62         top_ticket = ticket_system.peek()
63         if top_ticket:
64             print(f"Top ticket: {top_ticket}")
65
66         # Resolve all tickets in LIFO order
67         ticket_system.resolve_all()
68
69         # Display final stack state
70         print("\n" + "-" * 60)
71         print(f"Final Stack Size: {ticket_system.size()}")
72         print(f"Is Stack Empty? {ticket_system.isEmpty()}")
73         ticket_system.display_stack()
74
75 PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL POSTMAN CONSOLE
PS C:\AI Assistant Coding & C:\Users\edula\AppData\Local\Microsoft\Al\EndUser\python3.11.exe "c:\AI Assistant Coding\bug.py"
[3] Resolving next ticket...

```

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.

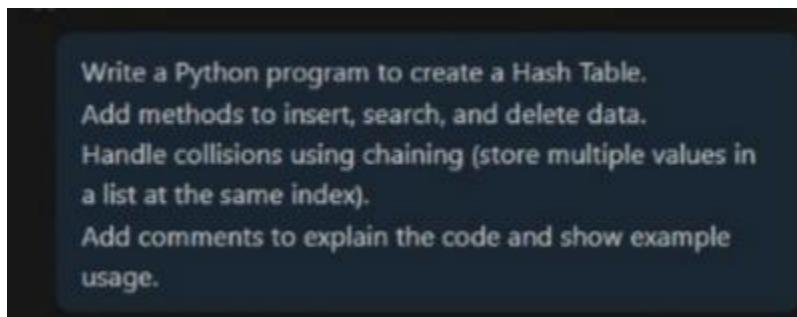
Task 4: Hash Table

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:

The screenshot shows a Microsoft Visual Studio Code (VS Code) window with the following details:

- Title Bar:** File, Edit, Selection, AI Assistant Coding.
- Sidebar:** Shows a tree view of files and folders, including Java, Python, and C/C++ files like `Assignment-7.java`, `Assignment-7.py`, and `Assignment-7.c`.
- Code Editor:** Displays Python code for a `HashTable` class. The code includes:
 - `insert(self, key, value)`: Inserts a key-value pair into the chain and updates self.total_items.
 - `search(self, key)`: Searches for a key in the hash table and returns the value or None if not found.
 - `delete(self, key)`: Deletes a key-value pair from the hash table and returns True if successful or False if key is not found.
 - `display(self)`: Prints the current state of the hash table.
- AI Assistant Coding Sidebar:** Shows suggestions for improving the code, such as "Handle collisions using chaining" and "Add comments to explain the code and show example usage".
- Status Bar:** Shows the path "C:\Users\Mehta\AppData\Local\Microsoft\AlAssitant\python\11.exe", the file name "AI Assistant Coding\log.py", and the status "PS Al Assistant Coding: C:\Users\Mehta\AppData\Local\Microsoft\AlAssitant\python\11.exe 'AI Assistant Coding\log.py'".

Output:

```

PS C:\VIAI Assistant Coding> & C:/Users/edula/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/VIAI Assistant Coding/bug.py"
Index 0: age: 30 -> city: New York -> department: IT
Index 1: name: Alice -> salary: 75000
Index 2: name: Bob -> 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 0)
✗ 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]
If another file is selected, the table will be cleared.
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
----- ALL REMAINING ITEMS -----
age: 31
department: IT
name: Alice
salary: 75000
phone: 555-1234
country: USA
Total items: 6
Is empty: False
PS C:\VIAI Assistant Coding>

```

Explanation:

- A Hash Table stores data using a key and value.
- A hash function decides where to store the data.
- Sometimes two keys go to the same place. This is called a collision.
- To solve collisions, we use chaining, meaning we store multiple items in a list at the same index.
- The program should allow adding, finding, and removing data correctly.

Task 5: Real-Time Application Challenge Scenario

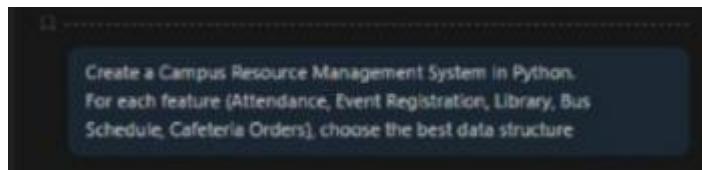
Design a Campus Resource Management System with the following features:

- Student Attendance Tracking

- Event Registration System
 - Library Book Borrowing
 - Bus Scheduling System
 - Cafeteria Order Queue

Prompt:

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



Code:

File Edit Selection ... ↶ ↷ 🔍 AI Assistant Coding

DIRECTORY - a

OPEN EDITORS

- J fb.java
- assypy
- evenoddsum.py
- student.py
- function.py
- temperATUREPY
- vote.py
- campus_resource_management.py
- bug.py
- library_book_request.py
- contact_m.py

AI ASSISTANT CODING

campus_resource_management.py

```

class Library:
    """Simple library using a dict for O(1) lookups by ISBN."""
    def __init__(self):
        self.catalog = {}

    def pending_count(self):
        return len(self.catalog)

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

    # ..... Library (dict/hashtable) .....
    class Library:
        """Simple 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(isbn) {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("User did not borrow the book.")
                return False
            book['copies'] += 1
            print(f'{user_id} returned {book["title"]}')
            return True

        def list_available(self):
            for (isbn, info), copies in self.catalog.items():
                print(f'{isbn}, {info["title"]}, {info["copies"]}')

```

bug.py

```

# Bus Schedule (route -> sorted list of times)
def bus_schedule(route):
    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 A', now + timedelta(minutes=25))
    print(f'Route A schedule: {bs.list_routes("Route A")}')
    print('Route A schedule:', bs.list_route('Route A'))

    # Cafeteria orders demo
    caf = CafeteriaOrders()
    caf.add_order('0001', '5001', ['Coffee', 'Sandwich'], customer_type='Student')
    caf.add_order('0002', 'F001', ['Salad'], customer_type='Faculty')
    caf.add_order('0003', 'F002', ['Pasta'], customer_type='Student')
    caf.add_order('0004', 'F003', ['Pasta'], customer_type='Faculty')
    print('Pending cafeteria orders:', caf.pending_count())
    caf.serve_order('0001')
    print('Pending orders after serving:', caf.pending_count())
    print('Order complete.')

```

It is not giving any output.

I'll open + bug.py to verify its current contents and the `bug()` 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 peak command!

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

Now Skip

Design data structures per feature (1/5)

1 file changed + 0 364

campus_resource_management.py

Describe what to build next

Lin 190 Col 20 Spaces 4 UTT-8 CR/LF Python 3.11.0 (Microsoft Store) # Go Live Previewer

File Edit Selection ... ↶ ↷ 🔍 AI Assistant Coding

DIRECTORY - a

OPEN EDITORS

- J fb.java
- assypy
- evenoddsum.py
- student.py
- function.py
- temperATUREPY
- vote.py
- campus_resource_management.py
- bug.py
- library_book_request.py
- contact_m.py

AI ASSISTANT CODING

campus_resource_management.py

```

# Attendance demo
att = Attendance()
att.mark_absent('5001')
att.mark_present('5002')
att.mark_present('5001')
print(f'{att.list_present()}')


# Event registration demo
ev = EventRegistration()
ev.add_registration('A001', 'Valent')
ev.add_registration('A002', 'Bob')
ev.add_registration('A003', 'Charlie')
print(f'{ev.list_pending()}')


# Library demo
lib = Library()
lib.add_book('978-0135166307', 'Clean Code', 'Robert C. Martin', copies=2)
lib.add_book('978-1683950826', 'Fluent Python', 'Luciano Ramalho', copies=1)
lib.borrow_book('978-0135166307', '5001')
lib.borrow_book('978-0135166307', '5002')
lib.borrow_book('978-0135166307', '5003') # no copies left
print(f'{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 A', now + timedelta(minutes=25))
print(f'Route A schedule: {bs.list_routes("Route A")}')
print('Route A schedule:', bs.list_route('Route A'))

# Cafeteria orders demo
caf = CafeteriaOrders()
caf.add_order('0001', '5001', ['Coffee', 'Sandwich'], customer_type='Student')
caf.add_order('0002', 'F001', ['Salad'], customer_type='Faculty')
caf.add_order('0003', 'F002', ['Pasta'], customer_type='Student')
caf.add_order('0004', 'F003', ['Pasta'], customer_type='Faculty')
print('Pending cafeteria orders:', caf.pending_count())
caf.serve_order('0001')
print('Pending orders after serving:', caf.pending_count())
print('Order complete.')

```

It is not giving any output.

I'll open + bug.py to verify its current contents and the `bug()` 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 peak command!

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

Now Skip

Design data structures per feature (1/5)

1 file changed + 0 364

campus_resource_management.py

Describe what to build next

Lin 190 Col 20 Spaces 4 UTT-4 CR/LF Python 3.11.0 (Microsoft Store) # Go Live Previewer

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: S003
Present list: ['S001', 'S002', 'S003']
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: 978-0135166387).
Added book: Fluent Python (ISBN: 978-1491958296).
S001 borrowed Clean Code
S003 borrowed Clean Code
No copies available.
Available books: [('978-0135166387', 'Clean Code', 0), ('978-1491958296', 'Fluent Python', 1)]
S001 returned Clean Code
Available books after return: [('978-0135166387', 'Clean Code', 1), ('978-1491958296', '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: 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.

