| | |
|---|---|
| **SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE** | **DEPARTMENT OF COMPUTER SCIENCE ENGINEERING** |

| | | |
|---|---|---|
| **Program Name:** B. Tech | **Assignment Type: Lab** | **Academic Year:**2025-2026 |

| | |
|---|---|
| **Course Coordinator Name** | Dr. Rishabh Mittal |
| **Instructor(s) Name** | Mr. S Naresh Kumar<br>Ms. B. Swathi<br>Dr. Sasanko Shekhar Gantayat<br>Mr. Md Sallauddin<br>Dr. Mathivanan<br>Mr. Y Srikanth<br>Ms. N Shilpa<br>Dr. Rishabh Mittal (Coordinator)<br>Dr. R. Prashant Kumar<br>Mr. Ankushavali MD<br>Mr. B Viswanath<br>Ms. Sujitha Reddy<br>Ms. A. Anitha<br>Ms. M.Madhuri<br>Ms. Katherashala Swetha<br>Ms. Velpula sumalatha<br>Mr. Bingi Raju |

| | | | |
|---|---|---|---|
| **Course Code** | 23CS002PC304 | **Course Title** | AI Assisted Coding |
| **Year/Sem** | III/II | **Regulation** | R23 |
| **Date and Day of Assignment** | **Week6 – Wednesday** | **Time(s)** | 23CSBTB01 To 23CSBTB52 |
| **Duration** | 2 Hours | **Applicable to Batches** | All batches |

**Assignment Number:11.3**(Present assignment number)/**24**(Total number of assignments)

| **Q.No.** | **Question** | ***Expected Time to complete*** |
|---|---|---|
| 1 | **Lab 11: Data Structures with AI Implementing Fundamental Data Structures using AI Assistance** | Week6 - |

Wednesday

**Lab Objectives:**

By the end of this lab, students will be able to:

- Design and implement fundamental data structures in Python using AI assistance.
- Effectively prompt AI tools (e.g., GitHub Copilot) for code generation, optimization, and documentation.
- Understand and compare core data structures: Arrays, Linked Lists, Stacks, Queues, Priority Queues, Trees, and Graphs.
- Improve code readability, efficiency, and maintainability using AI-generated suggestions.

**Learning Outcomes**

After completing this lab, students will be able to:

- Apply appropriate data structures to solve real-world problems.
- Analyze time and space complexity of different data structure operations.
- Use AI tools responsibly to assist (not replace) logical thinking and problem-solving.
- Validate, test, and refine AI-generated code.

---

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

**Scenario**

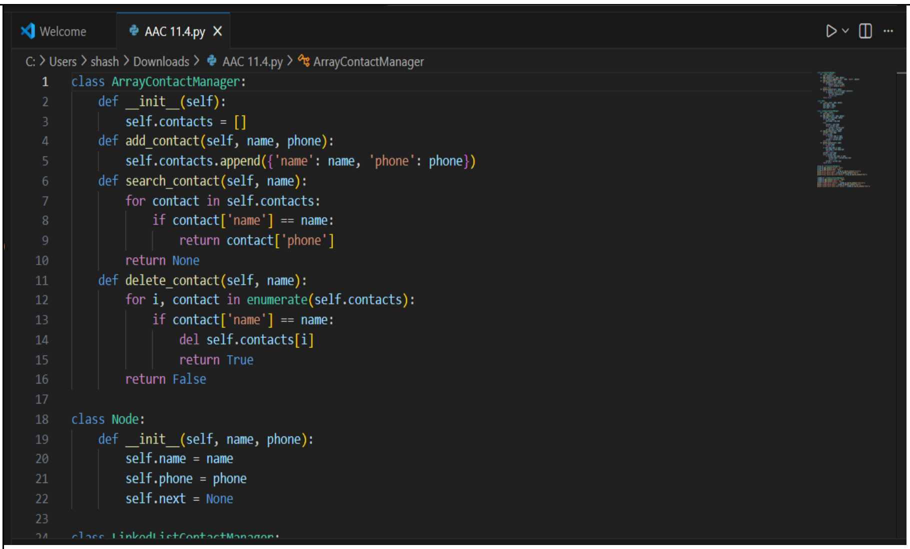
SR University’s student club requires a simple **Contact Manager Application** to store members’ names and phone numbers. The system should support efficient addition, searching, and deletion of contacts.

**Tasks**

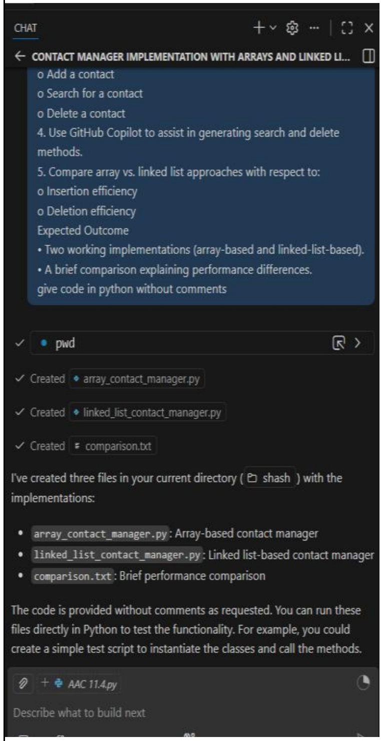
1. Implement the contact manager using **arrays (lists)**.
2. Implement the same functionality using a **linked list** for dynamic memory allocation.
3. Implement the following operations in both approaches:
    - Add a contact
    - Search for a contact
    - Delete a contact
4. Use **GitHub Copilot** to assist in generating search and delete methods.
5. Compare **array vs. linked list** approaches with respect to:
    - Insertion efficiency
    - Deletion efficiency

**Expected Outcome**

- Two working implementations (array-based and linked-list-based).
- A brief comparison explaining performance differences.


Welcome
AAC 11.4.py
C: > Users > shash > Downloads > AAC 11.4.py > ArrayContactManager

```python
class ArrayContactManager:
    def __init__(self):
        self.contacts = []
    def add_contact(self, name, phone):
        self.contacts.append({'name': name, 'phone': phone})
    def search_contact(self, name):
        for contact in self.contacts:
            if contact['name'] == name:
                return contact['phone']
        return None
    def delete_contact(self, name):
        for i, contact in enumerate(self.contacts):
            if contact['name'] == name:
                del self.contacts[i]
                return True
        return False

class Node:
    def __init__(self, name, phone):
        self.name = name
        self.phone = phone
        self.next = None

class LinkedListContactManager:
```



o Add a contact
o Search for a contact
o Delete a contact
4. Use GitHub Copilot to assist in generating search and delete methods.
5. Compare array vs. linked list approaches with respect to:
o Insertion efficiency
o Deletion efficiency
Expected Outcome
• Two working implementations (array-based and linked-list-based).
• A brief comparison explaining performance differences.
give code in python without comments

```
pwd
```

✓ Created array_contact_manager.py

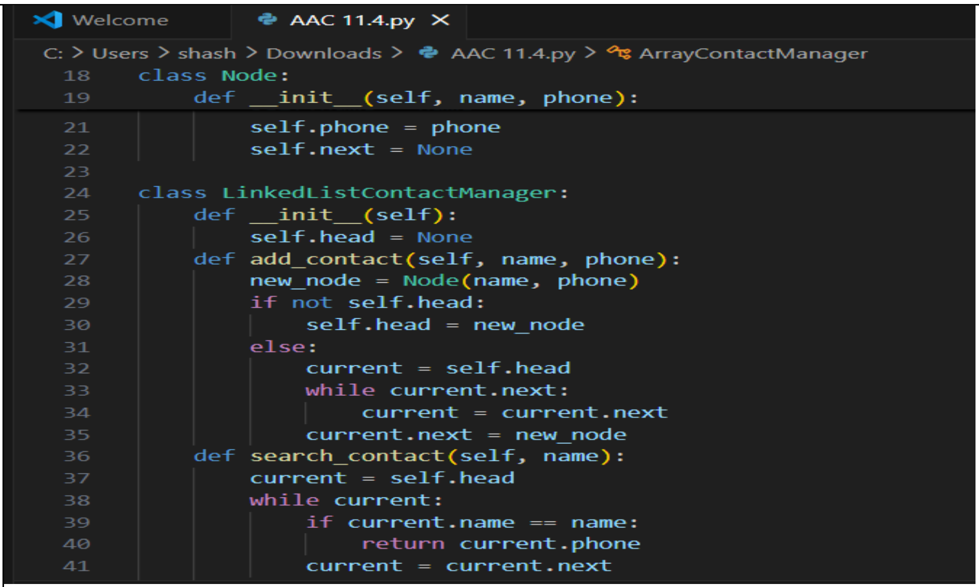
✓ Created linked_list_contact_manager.py

✓ Created comparison.txt

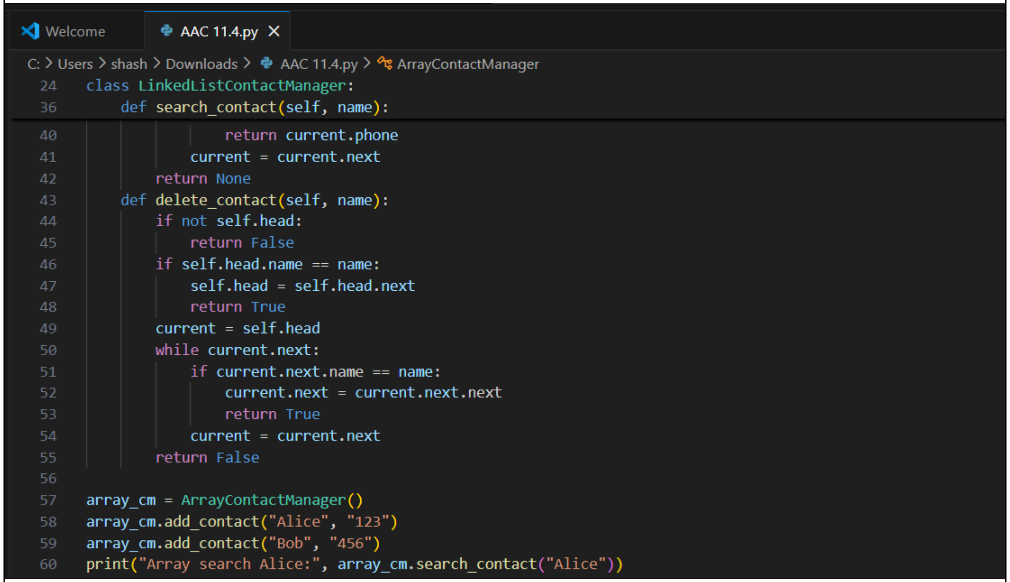
I've created three files in your current directory ( shash ) with the implementations:

- `array_contact_manager.py`: Array-based contact manager
- `linked_list_contact_manager.py`: Linked list-based contact manager
- `comparison.txt`: Brief performance comparison

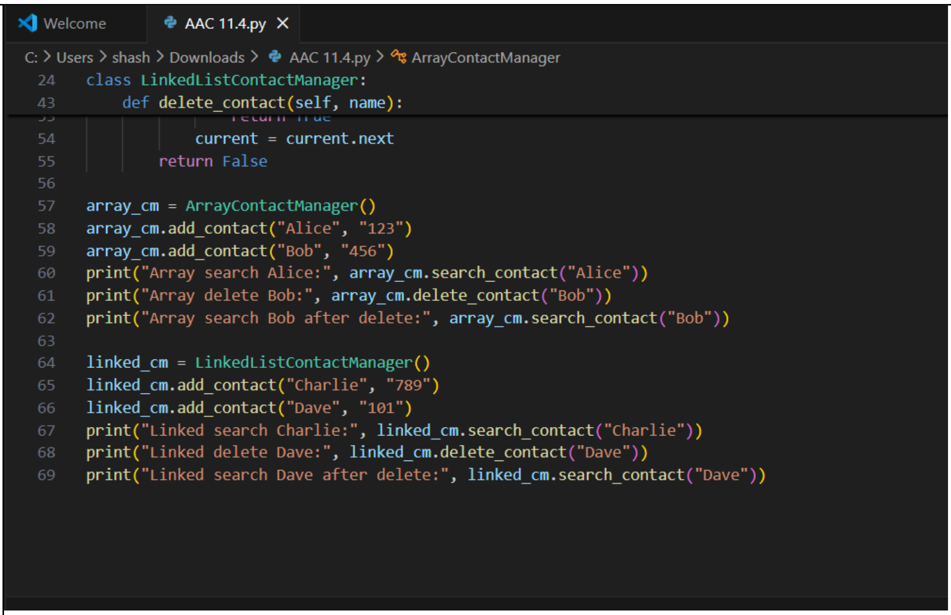
The code is provided without comments as requested. You can run these files directly in Python to test the functionality. For example, you could create a simple test script to instantiate the classes and call the methods.


Welcome
AAC 11.4.py
C: > Users > shash > Downloads > AAC 11.4.py > ArrayContactManager

```python
class Node:
    def __init__(self, name, phone):
        self.phone = phone
        self.next = None

class LinkedListContactManager:
    def __init__(self):
        self.head = None
    def add_contact(self, name, phone):
        new_node = Node(name, phone)
        if not self.head:
            self.head = new_node
        else:
            current = self.head
            while current.next:
                current = current.next
            current.next = new_node
    def search_contact(self, name):
        current = self.head
        while current:
            if current.name == name:
                return current.phone
            current = current.next
```


Welcome
AAC 11.4.py
C: > Users > shash > Downloads > AAC 11.4.py > ArrayContactManager

```python
class LinkedListContactManager:
    def search_contact(self, name):
                return current.phone
            current = current.next
        return None
    def delete_contact(self, name):
        if not self.head:
            return False
        if self.head.name == name:
            self.head = self.head.next
            return True
        current = self.head
        while current.next:
            if current.next.name == name:
                current.next = current.next.next
                return True
            current = current.next
        return False

array_cm = ArrayContactManager()
array_cm.add_contact("Alice", "123")
array_cm.add_contact("Bob", "456")
print("Array search Alice:", array_cm.search_contact("Alice"))
```


Welcome
AAC 11.4.py
C: > Users > shash > Downloads > AAC 11.4.py > ArrayContactManager

```python
class LinkedListContactManager:
    def delete_contact(self, name):
            current = current.next
        return False

array_cm = ArrayContactManager()
array_cm.add_contact("Alice", "123")
array_cm.add_contact("Bob", "456")
print("Array search Alice:", array_cm.search_contact("Alice"))
print("Array delete Bob:", array_cm.delete_contact("Bob"))
print("Array search Bob after delete:", array_cm.search_contact("Bob"))

linked_cm = LinkedListContactManager()
linked_cm.add_contact("Charlie", "789")
linked_cm.add_contact("Dave", "101")
print("Linked search Charlie:", linked_cm.search_contact("Charlie"))
print("Linked delete Dave:", linked_cm.delete_contact("Dave"))
print("Linked search Dave after delete:", linked_cm.search_contact("Dave"))
```

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

### Scenario

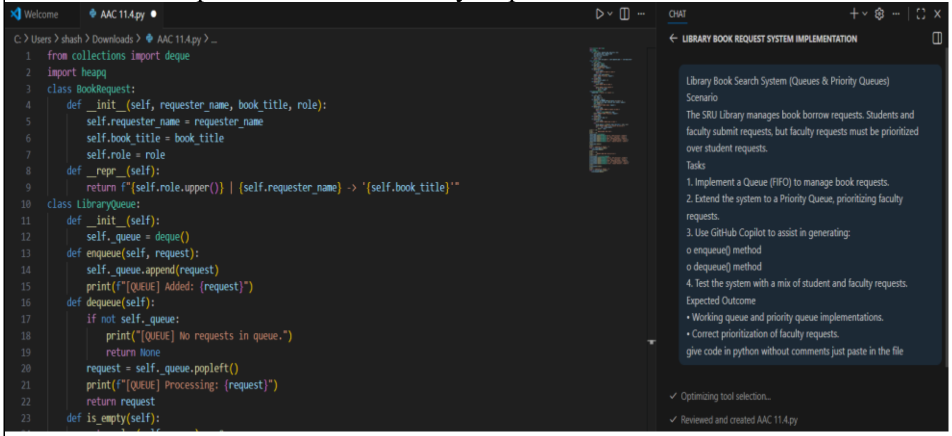
The SRU Library manages book borrow requests. Students and faculty submit requests, but **faculty requests must be prioritized** over student requests.

### Tasks

1. Implement a **Queue (FIFO)** to manage book requests.
2. Extend the system to a **Priority Queue**, prioritizing faculty requests.
3. Use **GitHub Copilot** to assist in generating:
    - enqueue() method
    - dequeue() method
4. Test the system with a mix of student and faculty requests.

### Expected Outcome

- Working queue and priority queue implementations.
- Correct prioritization of faculty requests.

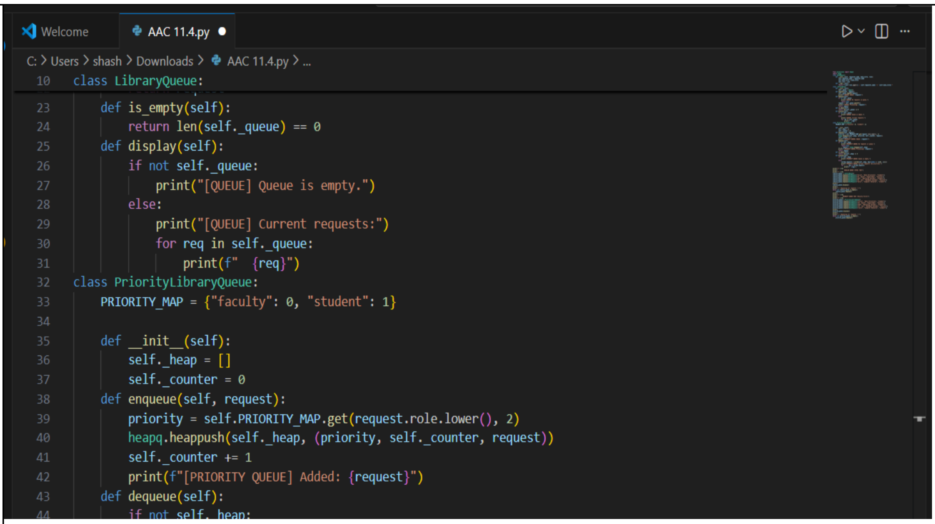

Welcome
AAC 11.4.py
C: > Users > shash > Downloads > AAC 11.4.py > ...
CHAT
LIBRARY BOOK REQUEST SYSTEM IMPLEMENTATION

```python
from collections import deque
import heapq
class BookRequest:
    def __init__(self, requester_name, book_title, role):
        self.requester_name = requester_name
        self.book_title = book_title
        self.role = role
    def __repr__(self):
        return f"{self.role.upper()} | {self.requester_name} -> '{self.book_title}'"
class LibraryQueue:
    def __init__(self):
        self._queue = deque()
    def enqueue(self, request):
        self._queue.append(request)
        print(f"[QUEUE] Added: {request}")
    def dequeue(self):
        if not self._queue:
            print("[QUEUE] No requests in queue.")
            return None
        request = self._queue.popleft()
        print(f"[QUEUE] Processing: {request}")
        return request
    def is_empty(self):
```

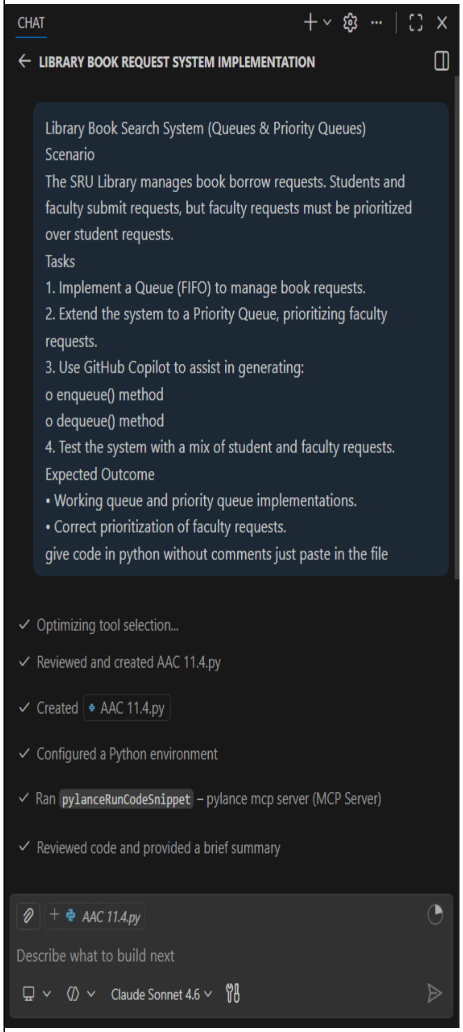
Library Book Search System (Queues & Priority Queues)
Scenario
The SRU Library manages book borrow requests. Students and faculty submit requests, but faculty requests must be prioritized over student requests.
Tasks
1. Implement a Queue (FIFO) to manage book requests.
2. Extend the system to a Priority Queue, prioritizing faculty requests.
3. Use GitHub Copilot to assist in generating:
o enqueue() method
o dequeue() method
4. Test the system with a mix of student and faculty requests.
Expected Outcome
• Working queue and priority queue implementations.
• Correct prioritization of faculty requests.
give code in python without comments just paste in the file

✓ Optimizing tool selection...

✓ Reviewed and created AAC 11.4.py



```python
class LibraryQueue:
    def is_empty(self):
        return len(self._queue) == 0
    def display(self):
        if not self._queue:
            print("[QUEUE] Queue is empty.")
        else:
            print("[QUEUE] Current requests:")
            for req in self._queue:
                print(f"  {req}")
class PriorityLibraryQueue:
    PRIORITY_MAP = {"faculty": 0, "student": 1}

    def __init__(self):
        self._heap = []
        self._counter = 0
    def enqueue(self, request):
        priority = self.PRIORITY_MAP.get(request.role.lower(), 2)
        heapq.heappush(self._heap, (priority, self._counter, request))
        self._counter += 1
        print(f"[PRIORITY QUEUE] Added: {request}")
    def dequeue(self):
        if not self._heap:
```



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.
Tasks
1. Implement a Queue (FIFO) to manage book requests.
2. Extend the system to a Priority Queue, prioritizing faculty requests.
3. Use GitHub Copilot to assist in generating:
o enqueue() method
o dequeue() method
4. Test the system with a mix of student and faculty requests.
Expected Outcome
• Working queue and priority queue implementations.
• Correct prioritization of faculty requests.
give code in python without comments just paste in the file

✓ Optimizing tool selection...

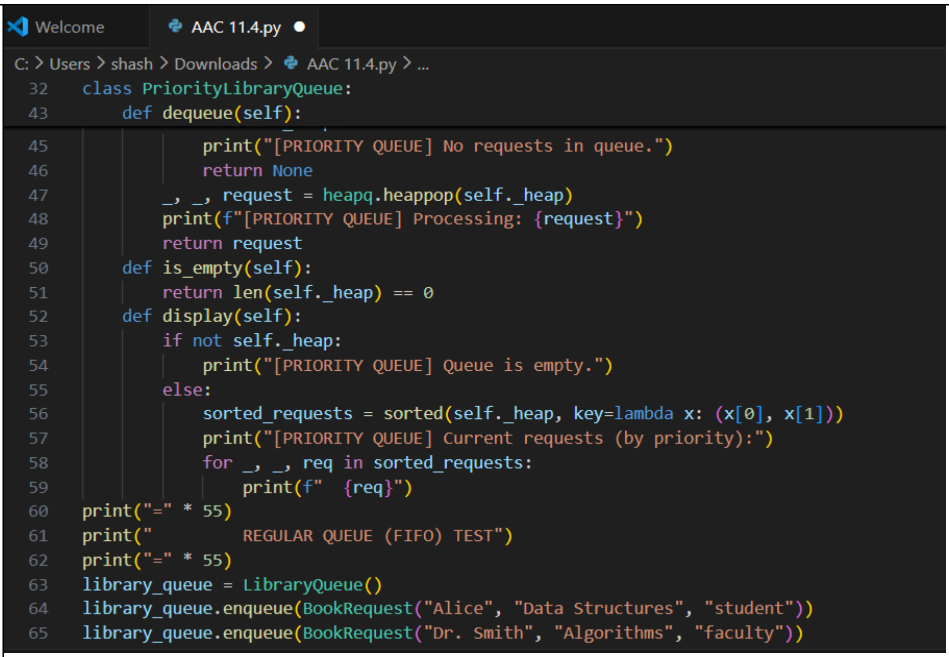
✓ Reviewed and created AAC 11.4.py

✓ Created AAC 11.4.py

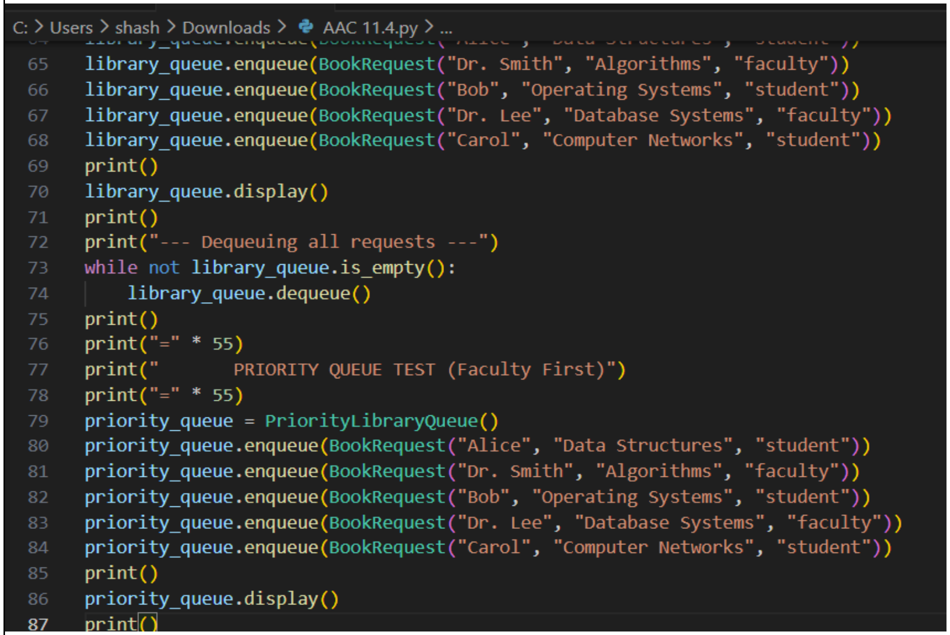
✓ Configured a Python environment

✓ Ran `pylanceRunCodeSnippet` – pylance mcp server (MCP Server)

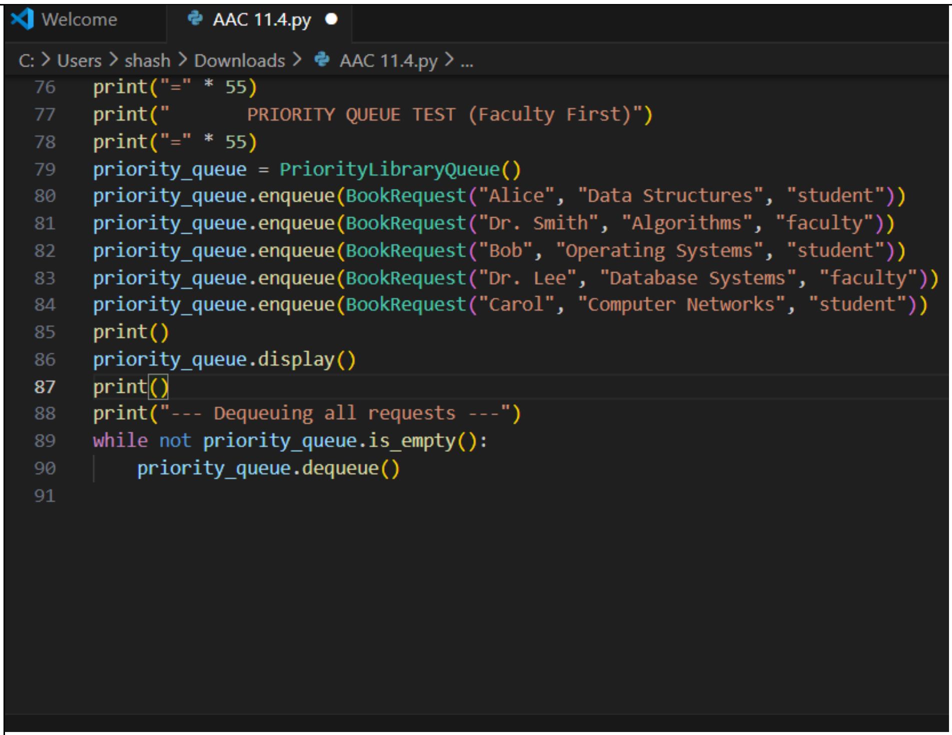
✓ Reviewed code and provided a brief summary



```python
class PriorityLibraryQueue:
    def dequeue(self):
            print("[PRIORITY QUEUE] No requests in queue.")
            return None
        _, _, request = heapq.heappop(self._heap)
        print(f"[PRIORITY QUEUE] Processing: {request}")
        return request
    def is_empty(self):
        return len(self._heap) == 0
    def display(self):
        if not self._heap:
            print("[PRIORITY QUEUE] Queue is empty.")
        else:
            sorted_requests = sorted(self._heap, key=lambda x: (x[0], x[1]))
            print("[PRIORITY QUEUE] Current requests (by priority):")
            for _, _, req in sorted_requests:
                print(f"  {req}")
print("=" * 55)
print("        REGULAR QUEUE (FIFO) TEST")
print("=" * 55)
library_queue = LibraryQueue()
library_queue.enqueue(BookRequest("Alice", "Data Structures", "student"))
library_queue.enqueue(BookRequest("Dr. Smith", "Algorithms", "faculty"))
```



```python
library_queue.enqueue(BookRequest("Dr. Smith", "Algorithms", "faculty"))
library_queue.enqueue(BookRequest("Bob", "Operating Systems", "student"))
library_queue.enqueue(BookRequest("Dr. Lee", "Database Systems", "faculty"))
library_queue.enqueue(BookRequest("Carol", "Computer Networks", "student"))
print()
library_queue.display()
print()
print("--- Dequeuing all requests ---")
while not library_queue.is_empty():
    library_queue.dequeue()
print()
print("=" * 55)
print("       PRIORITY QUEUE TEST (Faculty First)")
print("=" * 55)
priority_queue = PriorityLibraryQueue()
priority_queue.enqueue(BookRequest("Alice", "Data Structures", "student"))
priority_queue.enqueue(BookRequest("Dr. Smith", "Algorithms", "faculty"))
priority_queue.enqueue(BookRequest("Bob", "Operating Systems", "student"))
priority_queue.enqueue(BookRequest("Dr. Lee", "Database Systems", "faculty"))
priority_queue.enqueue(BookRequest("Carol", "Computer Networks", "student"))
print()
priority_queue.display()
print()
```



```python
print("=" * 55)
print("       PRIORITY QUEUE TEST (Faculty First)")
print("=" * 55)
priority_queue = PriorityLibraryQueue()
priority_queue.enqueue(BookRequest("Alice", "Data Structures", "student"))
priority_queue.enqueue(BookRequest("Dr. Smith", "Algorithms", "faculty"))
priority_queue.enqueue(BookRequest("Bob", "Operating Systems", "student"))
priority_queue.enqueue(BookRequest("Dr. Lee", "Database Systems", "faculty"))
priority_queue.enqueue(BookRequest("Carol", "Computer Networks", "student"))
print()
priority_queue.display()
print()
print("--- Dequeuing all requests ---")
while not priority_queue.is_empty():
    priority_queue.dequeue()

```

## Task 3: Emergency Help Desk (Stack Implementation)

### Scenario

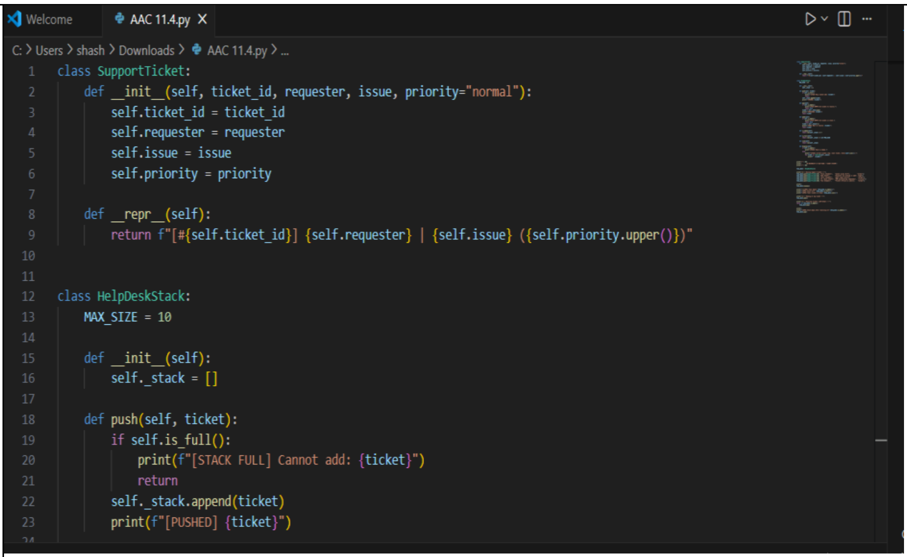
SR University’s IT Help Desk receives technical support tickets from students and staff. While tickets are received sequentially, issue escalation follows a **Last-In, First-Out (LIFO)** approach.

### Tasks

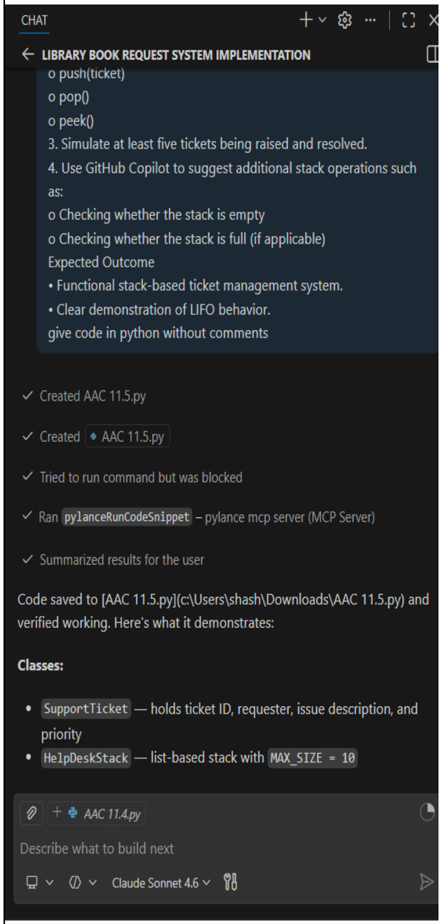
1. Implement a **Stack** to manage support tickets.
2. Provide the following operations:
    - push(ticket)
    - pop()
    - peek()
3. Simulate at least **five tickets** being raised and resolved.
4. Use **GitHub Copilot** to suggest additional stack operations such as:
    - Checking whether the stack is empty
    - Checking whether the stack is full (if applicable)

### Expected Outcome

- Functional stack-based ticket management system.
- Clear demonstration of LIFO behavior.



```python
class SupportTicket:
    def __init__(self, ticket_id, requester, issue, priority="normal"):
        self.ticket_id = ticket_id
        self.requester = requester
        self.issue = issue
        self.priority = priority

    def __repr__(self):
        return f"[#{self.ticket_id}] {self.requester} | {self.issue} ({self.priority.upper()})"


class HelpDeskStack:
    MAX_SIZE = 10

    def __init__(self):
        self._stack = []

    def push(self, ticket):
        if self.is_full():
            print(f"[STACK FULL] Cannot add: {ticket}")
            return
        self._stack.append(ticket)
        print(f"[PUSHED] {ticket}")
```



o push(ticket)
o pop()
o peek()
3. Simulate at least five tickets being raised and resolved.
4. Use GitHub Copilot to suggest additional stack operations such as:
o Checking whether the stack is empty
o Checking whether the stack is full (if applicable)
Expected Outcome
• Functional stack-based ticket management system.
• Clear demonstration of LIFO behavior.
give code in python without comments

✓ Created AAC 11.5.py

✓ Created AAC 11.5.py

✓ Tried to run command but was blocked

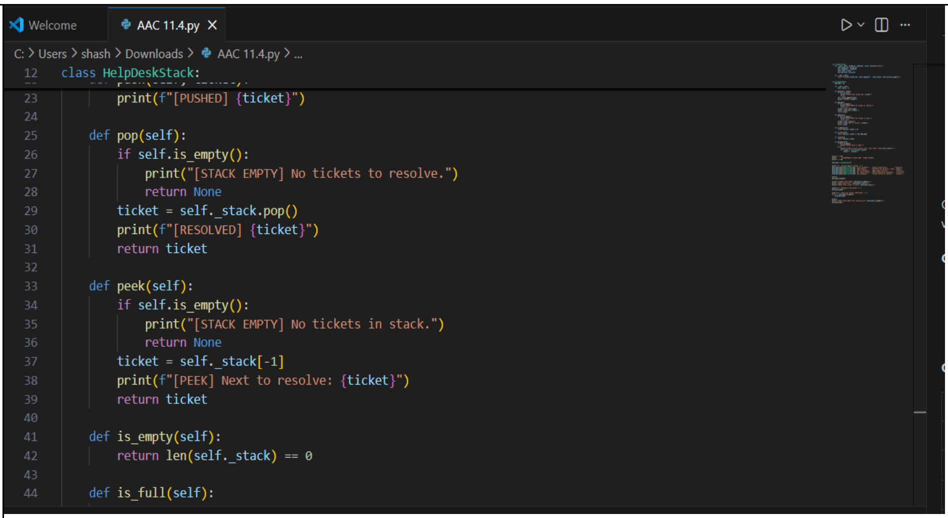
✓ Ran `pylanceRunCodeSnippet` – pylance mcp server (MCP Server)

✓ Summarized results for the user

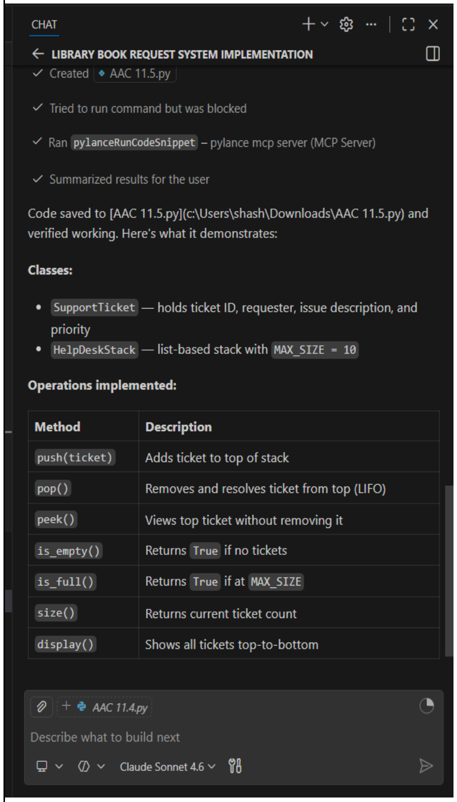
Code saved to [AAC 11.5.py](c:\Users\shash\Downloads\AAC 11.5.py) and verified working. Here's what it demonstrates:

**Classes:**

- `SupportTicket` — holds ticket ID, requester, issue description, and priority
- `HelpDeskStack` — list-based stack with `MAX_SIZE = 10`



```
class HelpDeskStack:
        print(f"[PUSHED] {ticket}")

    def pop(self):
        if self.is_empty():
            print("[STACK EMPTY] No tickets to resolve.")
            return None
        ticket = self._stack.pop()
        print(f"[RESOLVED] {ticket}")
        return ticket

    def peek(self):
        if self.is_empty():
            print("[STACK EMPTY] No tickets in stack.")
            return None
        ticket = self._stack[-1]
        print(f"[PEEK] Next to resolve: {ticket}")
        return ticket

    def is_empty(self):
        return len(self._stack) == 0

    def is_full(self):
```


CHAT
LIBRARY BOOK REQUEST SYSTEM IMPLEMENTATION
AAC 11.4.py
Describe what to build next
Claude Sonnet 4.6

✓ Created AAC 11.5.py

✓ Tried to run command but was blocked

✓ Ran `pylanceRunCodeSnippet` – pylance mcp server (MCP Server)

✓ Summarized results for the user

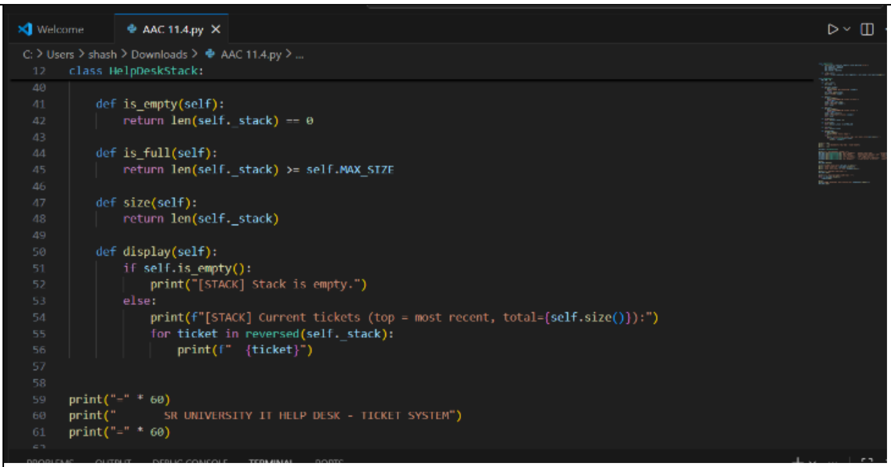
Code saved to [AAC 11.5.py](c:\Users\shash\Downloads\AAC 11.5.py) and verified working. Here's what it demonstrates:

**Classes:**

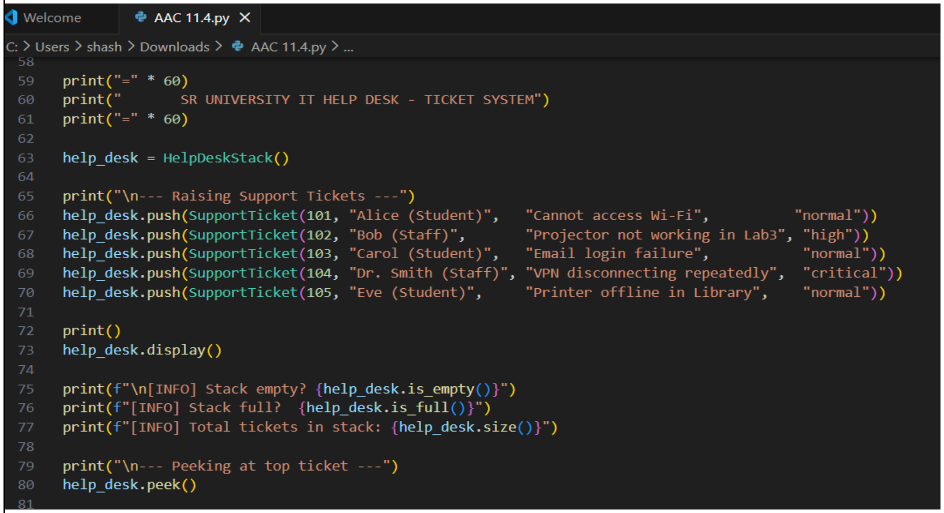
- `SupportTicket` — holds ticket ID, requester, issue description, and priority
- `HelpDeskStack` — list-based stack with `MAX_SIZE = 10`

**Operations implemented:**

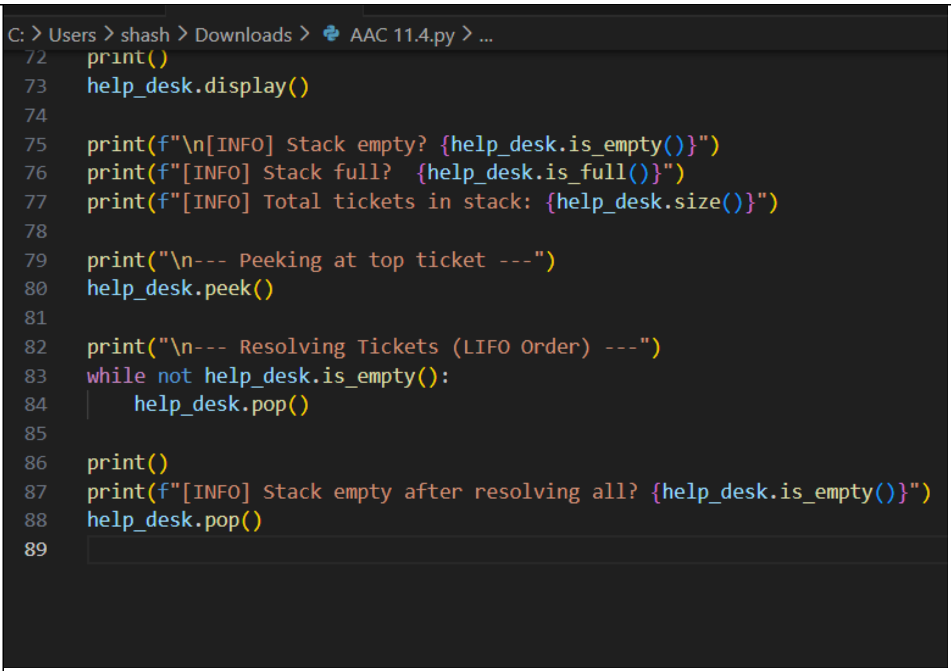
| **Method** | **Description** |
|---|---|
| `push(ticket)` | Adds ticket to top of stack |
| `pop()` | Removes and resolves ticket from top (LIFO) |
| `peek()` | Views top ticket without removing it |
| `is_empty()` | Returns `True` if no tickets |
| `is_full()` | Returns `True` if at `MAX_SIZE` |
| `size()` | Returns current ticket count |
| `display()` | Shows all tickets top-to-bottom |



```python
class HelpDeskStack:

    def is_empty(self):
        return len(self._stack) == 0

    def is_full(self):
        return len(self._stack) >= self.MAX_SIZE

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

    def display(self):
        if self.is_empty():
            print("[STACK] Stack is empty.")
        else:
            print(f"[STACK] Current tickets (top = most recent, total={self.size()}):")
            for ticket in reversed(self._stack):
                print(f"  {ticket}")


print("=" * 60)
print("       SR UNIVERSITY IT HELP DESK - TICKET SYSTEM")
print("=" * 60)
```



```python
print("=" * 60)
print("       SR UNIVERSITY IT HELP DESK - TICKET SYSTEM")
print("=" * 60)

help_desk = HelpDeskStack()

print("\n--- Raising Support Tickets ---")
help_desk.push(SupportTicket(101, "Alice (Student)",   "Cannot access Wi-Fi",          "normal"))
help_desk.push(SupportTicket(102, "Bob (Staff)",       "Projector not working in Lab3", "high"))
help_desk.push(SupportTicket(103, "Carol (Student)",   "Email login failure",           "normal"))
help_desk.push(SupportTicket(104, "Dr. Smith (Staff)", "VPN disconnecting repeatedly",  "critical"))
help_desk.push(SupportTicket(105, "Eve (Student)",     "Printer offline in Library",    "normal"))

print()
help_desk.display()

print(f"\n[INFO] Stack empty? {help_desk.is_empty()}")
print(f"[INFO] Stack full?  {help_desk.is_full()}")
print(f"[INFO] Total tickets in stack: {help_desk.size()}")

print("\n--- Peeking at top ticket ---")
help_desk.peek()
```



```python
print()
help_desk.display()

print(f"\n[INFO] Stack empty? {help_desk.is_empty()}")
print(f"[INFO] Stack full?  {help_desk.is_full()}")
print(f"[INFO] Total tickets in stack: {help_desk.size()}")

print("\n--- Peeking at top ticket ---")
help_desk.peek()

print("\n--- Resolving Tickets (LIFO Order) ---")
while not help_desk.is_empty():
    help_desk.pop()

print()
print(f"[INFO] Stack empty after resolving all? {help_desk.is_empty()}")
help_desk.pop()
```

## Task 4: Hash Table

### Objective

To implement a **Hash Table** and understand collision handling.

### Task Description

Use AI to generate a hash table with:

- Insert
- Search
- Delete

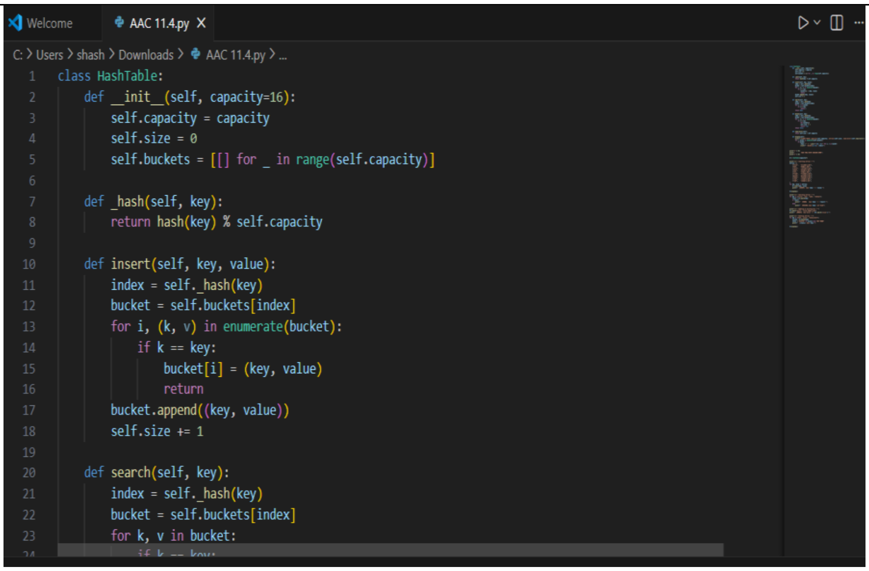
### Starter Code

class HashTable:

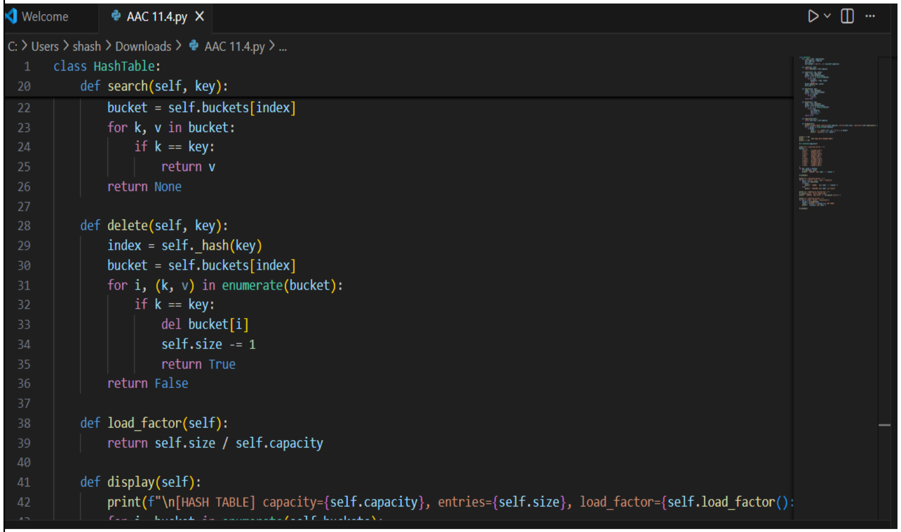
pass

### Expected Outcome

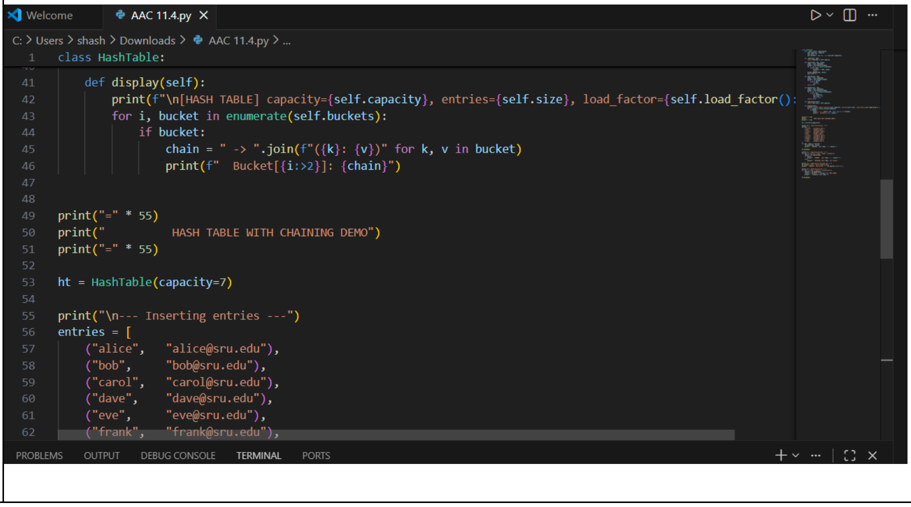
- Collision handling using chaining
- Well-commented methods



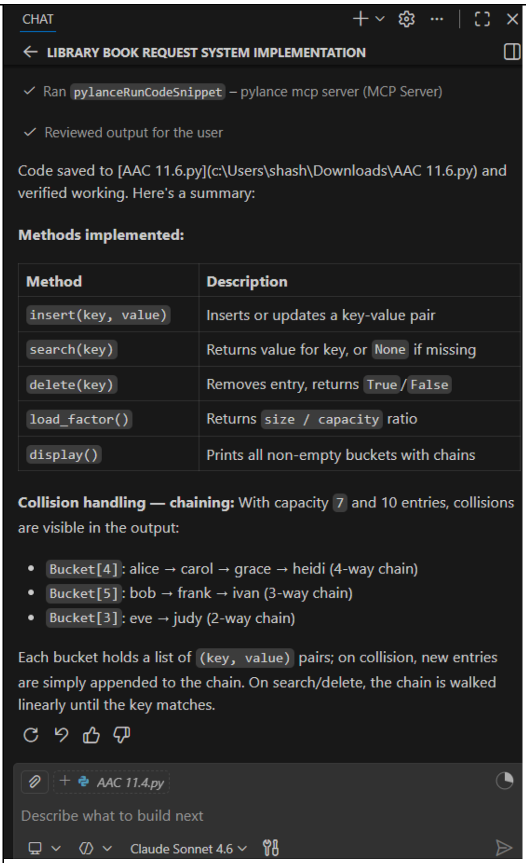
```python
class HashTable:
    def __init__(self, capacity=16):
        self.capacity = capacity
        self.size = 0
        self.buckets = [[] for _ in range(self.capacity)]

    def _hash(self, key):
        return hash(key) % self.capacity

    def insert(self, key, value):
        index = self._hash(key)
        bucket = self.buckets[index]
        for i, (k, v) in enumerate(bucket):
            if k == key:
                bucket[i] = (key, value)
                return
        bucket.append((key, value))
        self.size += 1

    def search(self, key):
        index = self._hash(key)
        bucket = self.buckets[index]
        for k, v in bucket:
            if k == key:
```



```python
class HashTable:
    def search(self, key):
        bucket = self.buckets[index]
        for k, v in bucket:
            if k == key:
                return v
        return None

    def delete(self, key):
        index = self._hash(key)
        bucket = self.buckets[index]
        for i, (k, v) in enumerate(bucket):
            if k == key:
                del bucket[i]
                self.size -= 1
                return True
        return False

    def load_factor(self):
        return self.size / self.capacity

    def display(self):
        print(f"\n[HASH TABLE] capacity={self.capacity}, entries={self.size}, load_factor={self.load_factor():
        for i, bucket in enumerate(self.buckets):
```



```python
class HashTable:
    def display(self):
        print(f"\n[HASH TABLE] capacity={self.capacity}, entries={self.size}, load_factor={self.load_factor():
        for i, bucket in enumerate(self.buckets):
            if bucket:
                chain = " -> ".join(f"({k}: {v})" for k, v in bucket)
                print(f"  Bucket[{i:>2}]: {chain}")


print("=" * 55)
print("          HASH TABLE WITH CHAINING DEMO")
print("=" * 55)

ht = HashTable(capacity=7)

print("\n--- Inserting entries ---")
entries = [
    ("alice",   "alice@sru.edu"),
    ("bob",     "bob@sru.edu"),
    ("carol",   "carol@sru.edu"),
    ("dave",    "dave@sru.edu"),
    ("eve",     "eve@sru.edu"),
    ("frank",   "frank@sru.edu"),
```



✓ Ran `pylanceRunCodeSnippet` – pylance mcp server (MCP Server)

✓ Reviewed output for the user

Code saved to [AAC 11.6.py](c:\Users\shash\Downloads\AAC 11.6.py) and verified working. Here's a summary:

**Methods implemented:**

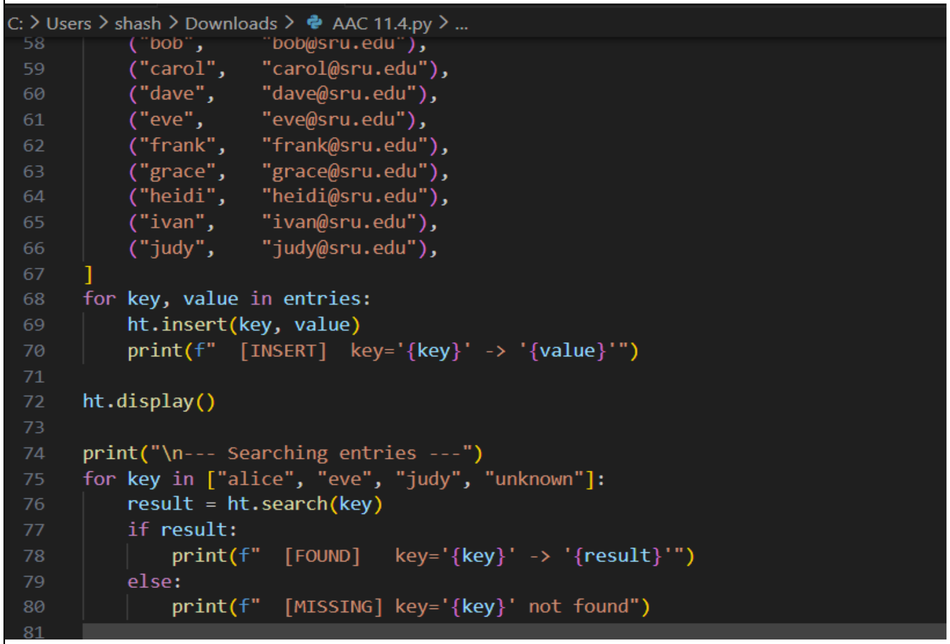
| **Method** | **Description** |
|---|---|
| `insert(key, value)` | Inserts or updates a key-value pair |
| `search(key)` | Returns value for key, or `None` if missing |
| `delete(key)` | Removes entry, returns `True`/`False` |
| `load_factor()` | Returns `size / capacity` ratio |
| `display()` | Prints all non-empty buckets with chains |

**Collision handling — chaining:** With capacity `7` and 10 entries, collisions are visible in the output:

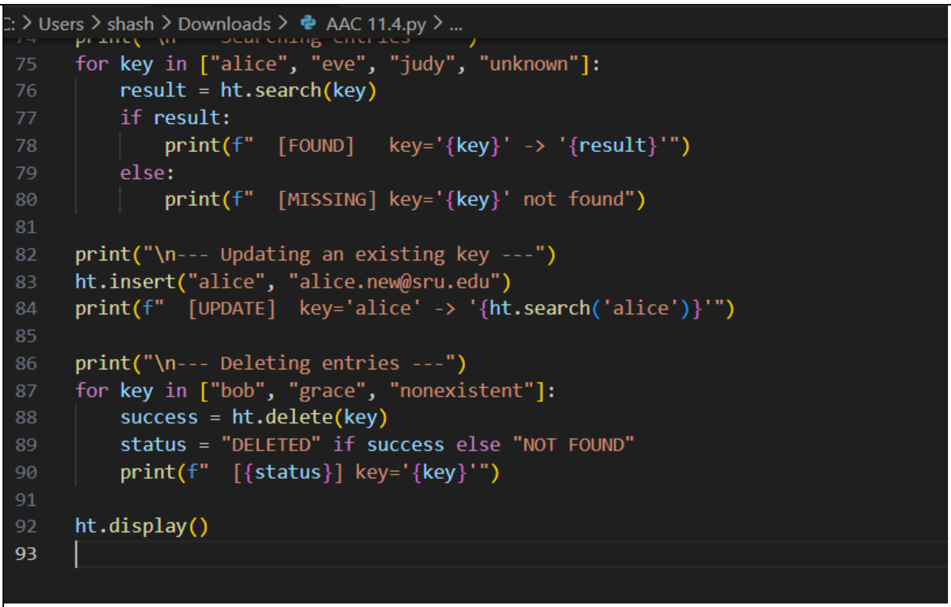
- `Bucket[4]`: alice → carol → grace → heidi (4-way chain)
- `Bucket[5]`: bob → frank → ivan (3-way chain)
- `Bucket[3]`: eve → judy (2-way chain)

Each bucket holds a list of `(key, value)` pairs; on collision, new entries are simply appended to the chain. On search/delete, the chain is walked linearly until the key matches.

AAC 11.4.py
Describe what to build next
Claude Sonnet 4.6


C: > Users > shash > Downloads > AAC 11.4.py > ...

```
        ("bob",     "bob@sru.edu"),
        ("carol",   "carol@sru.edu"),
        ("dave",    "dave@sru.edu"),
        ("eve",     "eve@sru.edu"),
        ("frank",   "frank@sru.edu"),
        ("grace",   "grace@sru.edu"),
        ("heidi",   "heidi@sru.edu"),
        ("ivan",    "ivan@sru.edu"),
        ("judy",    "judy@sru.edu"),
    ]
    for key, value in entries:
        ht.insert(key, value)
        print(f"  [INSERT]  key='{key}' -> '{value}'")

    ht.display()

    print("\n--- Searching entries ---")
    for key in ["alice", "eve", "judy", "unknown"]:
        result = ht.search(key)
        if result:
            print(f"  [FOUND]   key='{key}' -> '{result}'")
        else:
            print(f"  [MISSING] key='{key}' not found")

```



```python
print("\n--- Searching entries ---")
for key in ["alice", "eve", "judy", "unknown"]:
    result = ht.search(key)
    if result:
        print(f"  [FOUND]   key='{key}' -> '{result}'")
    else:
        print(f"  [MISSING] key='{key}' not found")

print("\n--- Updating an existing key ---")
ht.insert("alice", "alice.new@sru.edu")
print(f"  [UPDATE]  key='alice' -> '{ht.search('alice')}'")

print("\n--- Deleting entries ---")
for key in ["bob", "grace", "nonexistent"]:
    success = ht.delete(key)
    status = "DELETED" if success else "NOT FOUND"
    print(f"  [{status}] key='{key}'")

ht.display()

```

## Task 5: Real-Time Application Challenge

### Scenario

Design a **Campus Resource Management System** with the following features:

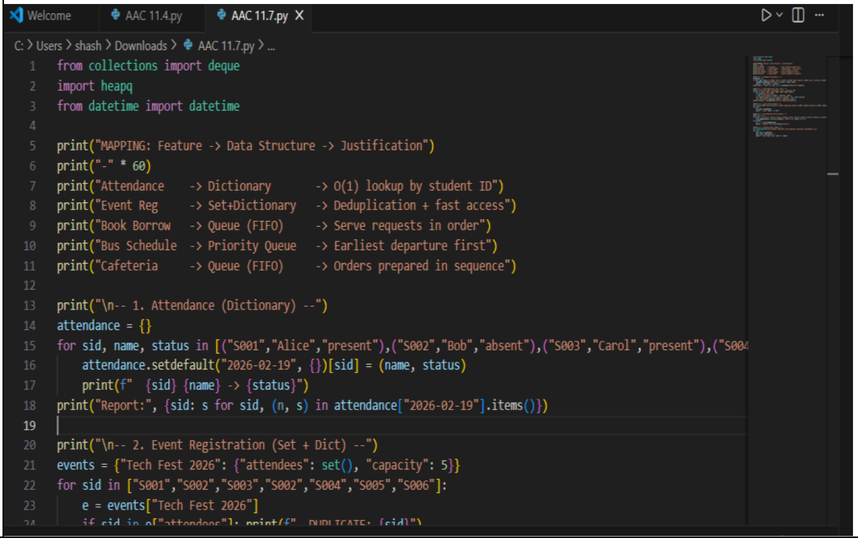
- Student Attendance Tracking
- Event Registration System
- Library Book Borrowing
- Bus Scheduling System
- Cafeteria Order Queue

### Student Tasks

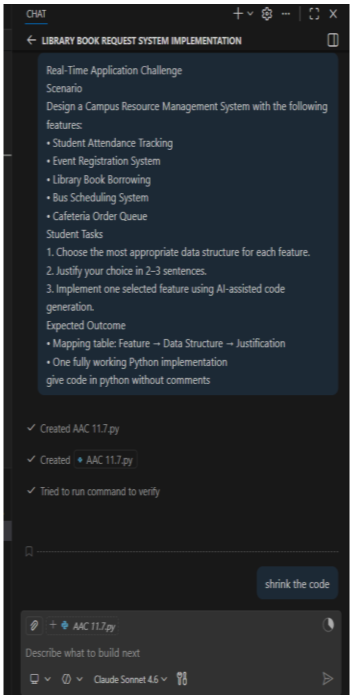
1. Choose the most appropriate data structure for each feature.
2. Justify your choice in **2–3 sentences**.
3. Implement **one selected feature** using AI-assisted code generation.

### Expected Outcome

- Mapping table: Feature → Data Structure → Justification
- One fully working Python implementation



```python
from collections import deque
import heapq
from datetime import datetime

print("MAPPING: Feature -> Data Structure -> Justification")
print("-" * 60)
print("Attendance    -> Dictionary       -> O(1) lookup by student ID")
print("Event Reg     -> Set+Dictionary   -> Deduplication + fast access")
print("Book Borrow   -> Queue (FIFO)     -> Serve requests in order")
print("Bus Schedule  -> Priority Queue   -> Earliest departure first")
print("Cafeteria     -> Queue (FIFO)     -> Orders prepared in sequence")

print("\n-- 1. Attendance (Dictionary) --")
attendance = {}
for sid, name, status in [("S001","Alice","present"),("S002","Bob","absent"),("S003","Carol","present"),("S004
    attendance.setdefault("2026-02-19", {})[sid] = (name, status)
    print(f"  {sid} {name} -> {status}")
print("Report:", {sid: s for sid, (n, s) in attendance["2026-02-19"].items()})

print("\n-- 2. Event Registration (Set + Dict) --")
events = {"Tech Fest 2026": {"attendees": set(), "capacity": 5}}
for sid in ["S001","S002","S003","S002","S004","S005","S006"]:
    e = events["Tech Fest 2026"]
    if sid in e["attendees"]: print(f"  DUPLICATE: {sid}")
```



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
Student Tasks
1. Choose the most appropriate data structure for each feature.
2. Justify your choice in 2–3 sentences.
3. Implement one selected feature using AI-assisted code generation.
Expected Outcome
• Mapping table: Feature → Data Structure → Justification
• One fully working Python implementation
give code in python without comments

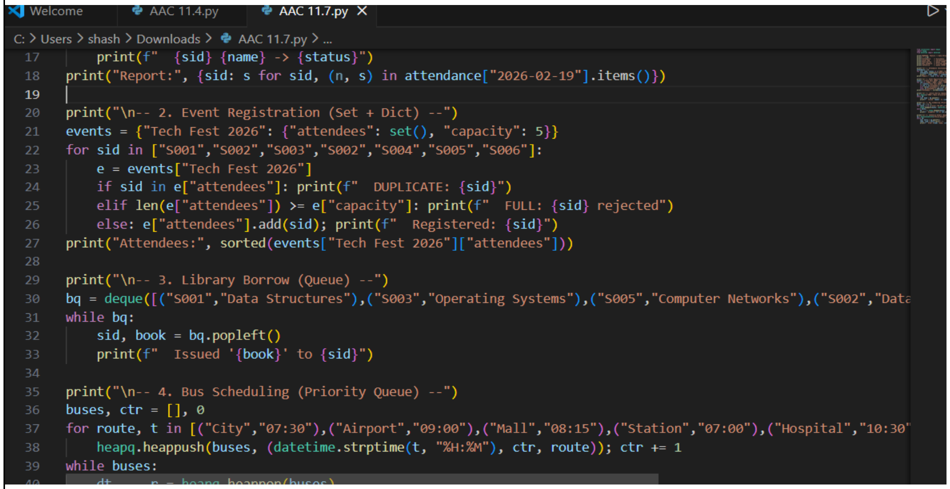
✓ Created AAC 11.7.py

✓ Created AAC 11.7.py

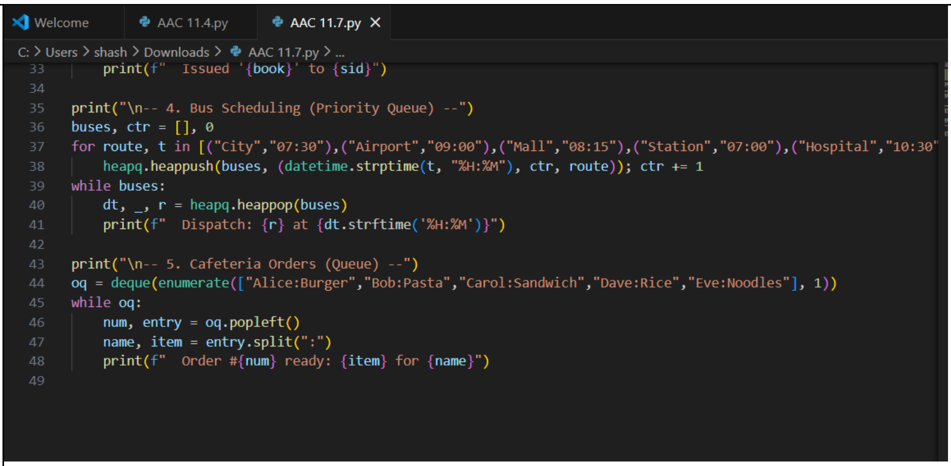
✓ Tried to run command to verify

shrink the code

AAC 11.7.py
Describe what to build next
Claude Sonnet 4.6


Welcome
AAC 11.4.py
AAC 11.7.py
C: > Users > shash > Downloads > AAC 11.7.py > ...

```python
    print(f"  {sid} {name} -> {status}")
print("Report:", {sid: s for sid, (n, s) in attendance["2026-02-19"].items()})

print("\n-- 2. Event Registration (Set + Dict) --")
events = {"Tech Fest 2026": {"attendees": set(), "capacity": 5}}
for sid in ["S001","S002","S003","S002","S004","S005","S006"]:
    e = events["Tech Fest 2026"]
    if sid in e["attendees"]: print(f"  DUPLICATE: {sid}")
    elif len(e["attendees"]) >= e["capacity"]: print(f"  FULL: {sid} rejected")
    else: e["attendees"].add(sid); print(f"  Registered: {sid}")
print("Attendees:", sorted(events["Tech Fest 2026"]["attendees"]))

print("\n-- 3. Library Borrow (Queue) --")
bq = deque([("S001","Data Structures"),("S003","Operating Systems"),("S005","Computer Networks"),("S002","Data
while bq:
    sid, book = bq.popleft()
    print(f"  Issued '{book}' to {sid}")

print("\n-- 4. Bus Scheduling (Priority Queue) --")
buses, ctr = [], 0
for route, t in [("City","07:30"),("Airport","09:00"),("Mall","08:15"),("Station","07:00"),("Hospital","10:30"
    heapq.heappush(buses, (datetime.strptime(t, "%H:%M"), ctr, route)); ctr += 1
while buses:
```



```python
    print(f"  Issued '{book}' to {sid}")

print("\n-- 4. Bus Scheduling (Priority Queue) --")
buses, ctr = [], 0
for route, t in [("City","07:30"),("Airport","09:00"),("Mall","08:15"),("Station","07:00"),("Hospital","10:30"
    heapq.heappush(buses, (datetime.strptime(t, "%H:%M"), ctr, route)); ctr += 1
while buses:
    dt, _, r = heapq.heappop(buses)
    print(f"  Dispatch: {r} at {dt.strftime('%H:%M')}")

print("\n-- 5. Cafeteria Orders (Queue) --")
oq = deque(enumerate(["Alice:Burger","Bob:Pasta","Carol:Sandwich","Dave:Rice","Eve:Noodles"], 1))
while oq:
    num, entry = oq.popleft()
    name, item = entry.split(":")
    print(f"  Order #{num} ready: {item} for {name}")
```

Note: Report should be submitted as a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots.