

Assignment 11.3 Ai Assisted Coding

Ht.no: 2303A51305

Batch: 05

Task 1:

Smart Contact Manager (Arrays & Linked Lists)

Scenario

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

Prompt:

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

Code:

The screenshot shows a Windows desktop environment with a code editor window open. The code editor displays Python code for a contact manager. A sidebar titled "Code Review Assistant" provides analysis and suggestions for the code.

Code Review Assistant Analysis:

- Overall:** The code is well-organized and follows good practices.
- Imports:** Imports are present at the top of the file.
- Variables:** Variables are clearly named.
- Functions:** Functions are well-defined and serve specific purposes.
- Comments:** Comments are included throughout the code to explain logic.
- Style:** The code uses standard Python conventions (e.g., snake_case for variable names).
- Performance:** The code includes a linked list implementation, which is efficient for certain operations like insertion and deletion.
- Testing:** No explicit tests or assertions are shown in the provided code.

Code Review Assistant Screenshot:

- Top Bar:** Shows navigation icons (File, Edit, Selection, etc.) and a search bar.
- Code Editor:** Displays the Python contact manager code.
- Code Review Sidebar:**
 - Code Review Assistant:** A large section showing analysis results.
 - Issues:** A list of potential issues or suggestions.
 - Code Metrics:** Statistics about the code (LOC, complexity, etc.).
 - Code Coverage:** Coverage analysis for different parts of the code.
 - Code Operations:** A section for managing code operations.
 - Demo:** A section for demonstrating code features.

Output:

```

PS C:\AI Assistant Coding & C:/Users/edula/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/AI Assistant Coding/bug.py"
diana@email.com
No matching results
powershell
powershell
Python
BUG FIXING ASSISTANT

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

Planned Contact Manager program with lists and linked lists.
I'll create a comprehensive Contact Manager program with both list and linked list implementations.

Created contact_manager.py
cd
cd\AI Assistant Coding\;
&
python contact_manager.py

Run contact manager program and demonstrated all features.

Perfect! I've created a comprehensive Contact Manager program saved to contact_manager.py with the following features.

Classes Included:
contact_manager.py

Describe what to build next
Python 3.11.9 (Microsoft Store)
1 Hidden ...
In 226, Col 1 Spaces: 4 UTF-8 Python 3.11.9 (Microsoft Store) Go Live Prettier

```

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:

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

- Title Bar:** AI Assistant Coding
- Left Sidebar (Explorer):** Shows the file structure and open files. Files include: fb.java, assypy, evenoddsum.py, student.py, function.py, temPERATURE.PY, vote.py, bus.py, library_book_request.py, contact_manager.py, another_file.txt, ASSIGNMENT_3-3.docx, ASSIGNMENT1-3.docx, ASSIGNMENT2-3.docx, ASSIGNMENT4-4.docx, ASSIGNMENT4-1.docx, Assignment-7-3.docx, assypy, assypy, contact_manager.py, evenoddsum.py, Jfb.class, Jfb.java, file.py, function.py, library_book_request.py, simpleif, student.py, sumpy, temPERATURE.PY, vote.py.
- Central Area (Editor):** The main editor area displays Python code for a Priority Queue system. The code includes methods for adding requests, removing requests, displaying the queue, and processing requests. It also includes a section for a Normal Queue demo and a section for a Priority Queue demo.
- Right Sidebar (Output):** Shows the output of the code execution, including logs and command-line interface interactions.

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

- Title Bar:** AI Assistant Coding
- File Explorer (Left):** Shows files like fb.java, asupy, evenoddsum.py, student.py, function.py, temperature.py, vote.py, and buggy.
- Code Editor (Main):** Displays Python code for implementing different queue types and processing requests. The code includes functions for adding requests to queues (normal_q, priority_q), removing requests, and displaying queue sizes.
- Output Panel (Bottom Right):** Shows logs and command-line outputs related to the code execution.
- Status Bar (Bottom):** Shows file paths (fb.java, asupy, etc.), line numbers (e.g., L 43 Col 20), and other system information.

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.

Code:

This screenshot shows the Microsoft Visual Studio Code interface with the "AI Assistant Coding" extension installed. The top bar includes tabs for File, Edit, Selection, and AI Assistant Coding. The Explorer sidebar shows a tree view of files, with "bug.py" currently selected. The main editor area contains the code for "bug.py", which is a stack-based ticket management system. The status bar at the bottom indicates the file is 100% Resolved and has 87 lines of code. A floating panel on the right provides "BUG FIXING ASSISTANCE" and a "REVIEWED AND UPDATED bug.py" section. The bottom right corner features a "Describe what to build next" input field.

```
 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
13  class Stack():
14      """Stack-based ticket management system (LIFO - Last In, First Out)"""
15      def __init__(self, max_size=10):
16          self.stack = []
17          self.max_size = max_size
18
19      def push(self, ticket):
20          """Add a ticket to the stack (top of stack)"""
21          if self.isFull():
22              print("Error: Stack is Full! Cannot add ticket #{}(ticket.ticket_id)")
23              return False
24
25          self.stack.append(ticket)
26          print(f"Ticket added: {ticket}")
27          return True
28
29      def pop(self):
30          """Remove and return the ticket from the top of the stack"""
31          if self.isEmpty():
32              print("Error: Stack is empty! No tickets to resolve.")
33              return None
34
35          ticket = self.stack.pop()
36          print(f"Resolving: {ticket}")
37          return ticket
38
39      def peek(self):
40          """View the top ticket without removing it"""
41          if self.isEmpty():
42              print("Error: Stack is empty!")
43              return None
44          return self.stack[-1]
45
46      def isEmpty(self):
47          """Check if the stack is empty"""
48          return len(self.stack) == 0
49
50      def isFull(self):
51          """Check if the stack is full"""
52          return len(self.stack) > self.max_size
53
54      def size(self):
55          """Return the number of tickets in the stack"""
56          return len(self.stack)
57
58      def display_stack(self):
59          """Display all tickets in the stack (top to bottom)"""
60
```

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

- File Explorer:** Shows multiple Python files: `fb.java`, `assis.py`, `evenoddsum.py`, `student.py`, `function.py`, `temperature.py`, `vote.py`, `bug.py`, `library_book_request.py`, and `contact_manager.py`. A file named `ASSIGNMENT CODING` is expanded, showing sub-files `2305A15182_ASSIGNN...` through `2305A15182_ASSIGNN...`.
- Editor:** The main editor window displays the `bug.py` file. The code implements a stack-based ticket system. It includes a `main` function that pushes five tickets onto the stack and then iterates to resolve them. Each ticket is a tuple containing a name and a status message.
- Terminal:** The bottom terminal shows the command `cd "C:\AI Assistant Coding" & python bug.py` being run.
- Output:** The output pane shows the execution results, including the initial stack state and the resolved ticket details.

Output:

```

PS C:\AI Assistant Coding & C:/Users/eduka/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/AI Assistant Coding/bug.py"

[1] Resolving next ticket...
✓ Resolving Ticket #101 | Customer: John Smith | Issue: Login issue
✓ All tickets have been resolved!
-----
Final Stack Size: 0
Is Stack Empty? True
Stack is empty! No tickets to display.

PS C:\AI Assistant Coding>

[2] Resolving next ticket...
✓ Resolving Ticket #100 | Customer: Emma Brown | Issue: Account suspended
[3] Resolving next ticket...
✓ Resolving Ticket #100 | Customer: Mike Wilson | Issue: Data export failed
[4] Resolving next ticket...
✓ Resolving Ticket #99 | Customer: Sarah Johnson | Issue: Payment processing error
[5] Resolving next ticket...
✓ Resolving Ticket #98 | Customer: John Smith | Issue: Login issue
All tickets have been resolved!
-----
[2] Resolving next ticket...
✓ Resolving Ticket #98 | Customer: Emma Brown | Issue: Account suspended
[3] Resolving next ticket...
✓ Resolving Ticket #98 | Customer: Sarah Johnson | Issue: Payment processing error
[4] Resolving next ticket...
✓ Resolving Ticket #97 | Customer: Mike Wilson | Issue: Data export failed
[5] Resolving next ticket...
✓ Resolving Ticket #96 | Customer: John Smith | Issue: Login issue
[1] Resolving next ticket...
✓ Resolving Ticket #96 | Customer: Mike Wilson | Issue: Data export failed
[4] Resolving next ticket...
✓ Resolving Ticket #96 | Customer: Sarah Johnson | Issue: Payment processing error
[5] Resolving next ticket...
[4] Resolving next ticket...
[4] Resolving Ticket #95 | Customer: Sarah Johnson | Issue: Payment processing error
[5] Resolving next ticket...
[4] Resolving Ticket #94 | Customer: John Smith | Issue: Login issue
All tickets have been resolved!
-----
Final Stack Size: 0
Is Stack Empty? True
Stack is empty! No tickets to display.

PS C:\AI Assistant Coding>

```

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) interface with the following details:

- Title Bar:** File, Edit, Selection, ..., AI Assistant Coding
- Left Sidebar (File Explorer):** Shows a tree view of files and folders, including `lucky.py`, `library_book_reque...`, and `contact_manager.py`.
- Central Area:** Displays Python code for a Hash Table with Chaining Collision Handling. The code defines a `KeyValuePair` class and a `HashTable` class. The `HashTable` class uses chaining to handle collisions, storing multiple key-value pairs that hash to the same index in a linked list.
- Right Sidebar (AI Assistant Coding):** Provides suggestions for improving the code:
 - Write a Python program to create a Hash Table.
 - Add methods to insert, search, and delete.
 - Handle collisions using chaining (store multiple values in a list at the same index).
 - Add comments to explain the code and show example usage.
- Status Bar:** Shows the file is open in "Python" mode, with other tabs like "HTML" and "JavaScript" visible.

The screenshot shows a Python code editor interface with the following details:

- Title Bar:** AI Assistant Coding
- File Explorer:** Shows files like `assigment_7.py`, `library_book_repositroy.py`, and `contact_manager.py`.
- Code Editor:** Displays a `hashTable.py` file containing a Hash Table implementation using chaining.
- Code Completion/Assistance:** The right side of the screen shows AI-generated code snippets and explanations for methods like `insert`, `search`, `delete`, and `display`.
- Terminal:** Shows the command `cd "C:\AI Assistant Coding" & python bug.py` being run.
- Status Bar:** Includes file paths, line numbers, and other development information.

Output:

This screenshot shows the Microsoft Visual Studio Code interface with the 'AI Assistant Coding' extension installed. The top bar includes tabs for File, Edit, Selection, Problems, Output, Debug Console, Ports, Terminal, and Postman Console. A search bar at the top right contains the text 'AI Assistant Coding'. The left sidebar has sections for Explorer, Problems, Output, Debug Console, Ports, Terminal, and Postman Console. The main editor area displays a series of command-line outputs from the terminal:

- File operations: j.java, ass.py, evenoddsum.py, function.py, temperature.py, voltage.py.
- Terminal output:
 - Index 1: age: 30 -> city: New York -> department: IT
 - Index 2: name: Alice -> salary: 75000
 - Index 3: phone: 555-1234 -> country: USA
 - Index 4: (empty)
- Total items in hash table: 8
- SEARCHING FOR DATA ...
 - YFound: email: alice@email.com (at index 2)
 - Y Found: age = 30 (at index 1)
 - X Key "unknown_key" not found in the hash table
- UPDATING DATA ...
 - YUpdated: 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)
- Total items in hash table: 8
- DELETING DATA ...
 - YDeleted: email = alice@email.com (from index 0)
 - YDeleted: city = New York (from index 1)
 - X Key "nonexistent" not found in the hash table
- HASH TABLE CONTENTS
 - Index 0: (empty)
 - Index 1: age: 31 -> department: IT
 - Index 2: name: Alice -> salary: 75000
 - Index 3: phone: 555-1234 -> country: USA
 - Index 4: (empty)
- Total items in hash table: 6
- ALL REMAINING ITEMS ...
 - age: 31
 - department: IT
 - name: Alice
 - salary: 75000
 - phone: 555-1234
 - country: USA
- Total items: 6
- Is this correct? False
- PS C:\AI Assistant Coding\]

The right side of the interface features a Chat window titled 'BUG FIXING ASSISTANCE system' with a message about fixing a bug in the code. Below the Chat is a 'Run in terminal' section with a 'Run in terminal' button and a 'Run in command' button. At the bottom, there's a status bar with file paths like 'C:\Users\adala\appdata\local\Microsoft\WindowsApps\python11.exe', a progress bar showing '1 / 1000 T...', and a '100% Auto' setting.

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:

```
campus_resource_management.py
1 #!/usr/bin/python
2
3 # Features and chosen data structures:
4 # - Attendance: set (O(1) add/check) to track present student IDs
5 # - Event Registration: FIFO queue to register attendees in arrival order
6 # - Library Book Borrowing: dict (hash table using ISBN) -> book record for fast lookup
7 # - Bus Schedule: dict of route -> sorted list of departure times (list kept sorted)
8 # - Cafeteria Orders: heapq (priority queue) to prioritize faculty over students while preserving arrival order
9
10 Run this file to see a small demo of each feature.
11
12
13 from collections import deque
14 import heapq
15 import itertools
16
17 from bisect import insert
18 from datetime import datetime, timedelta
19
20 # ..... Attendance (set) .....
21 class Attendance:
22     """Check attendance using a set for O(1) add/remove/check."""
23     def __init__(self):
24         self.present = set()
25
26     def mark_present(self, student_id):
27         self.present.add(student_id)
28         print("Marked present (%s)" % student_id)
29
30     def mark_absent(self, student_id):
31         self.present.discard(student_id)
32         print("Marked absent (%s)" % student_id)
33
34     def is_present(self, student_id):
35         return student_id in self.present
36
37     def present_count(self):
38         return len(self.present)
39
40     def list_present(self):
41         return sorted(self.present)
42
43 # ..... Event Registration (FIFO queue) .....
44 class EventRegistration:
45     """Register attendees in arrival order using deque."""
46     def __init__(self):
47         self.queue = deque()
48
49     def add_registration(self, attendee_id, name):
50         self.queue.append((attendee_id, name))
51         print("Registered (%s) - (%s)" % (attendee_id, name))
52
53     def process_registration(self):
54         if not self.queue:
55             print("No registrations to process.")
56         else:
57             attendee = self.queue.popleft()
58             print("Processed registration: (%s) - (%s)" % (attendee[0], attendee[1]))
59             return attendee
60
61     def pending_count(self):
62         return len(self.queue)
```

File Edit Selection ...

AI Assistant Coding

EXPLORER

OPEN EDITORS

- J fbclass
- assypy
- evenoddsum.py
- studentpy
- function.py
- temperaturePY
- vote.py
- campus_resource_management.py
- bug.py
- library_book_request.py
- contact_m

CHAT

BUG FIXING ASSISTANCE

```

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 copies 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
        if book['copies'] == 1:
            book['borrowers'].append(user_id)
            print(f"User {user_id} borrowed '{book['title']}'.")
            return True
        book['copies'] -= 1
        print(f"User {user_id} borrowed '{book['title']}'.")
        return True
    def list_available(self):
        return [(isbn, info['title'], info['copies']) for isbn, info in self.catalog.items()]

```

It is not giving any output.

I'll open + bug.py to verify its current contents and the `main` 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 `psql` command:

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

More Skip

Design data structures per feature (1/5)

1 file changed +103 -364

campus_resource_management.py

Describe what to build next

Ln 190 Col 29 Spaces: 4 UFT-8 CPU Python 3.11.0 (Microsoft Store) Go Live Preview

File Edit Selection ...

AI Assistant Coding

EXPLORER

OPEN EDITORS

- J fbclass
- assypy
- evenoddsum.py
- studentpy
- function.py
- temperaturePY
- vote.py
- campus_resource_management.py
- bug.py
- library_book_request.py
- contact_m

CHAT

BUG FIXING ASSISTANCE

```

if __name__ == '__main__':
    print("Campus Resource Management Demo")
    print("-----")
    # Attendance demo
    att = Attendance()
    att.mark_absent('50001')
    att.mark_present('50002')
    att.mark_present('50010')
    print("Present list: ", att.list_present())
    print("Absent list: ", att.is_present('5002'))
    att.mark_absent('5002')
    print("Present count: ", att.present_count())
    # Event registration demo
    ev = EventRegistration()
    ev.add_registration("Alice", "Robert C. Martin", copies=2)
    ev.add_registration("Bob", "Fluent Python", copies=1)
    ev.add_registration("Alice", "Charlie")
    print("Pending registrations: ", ev.list_pending())
    ev.process_registration()
    print("Pending count: ", ev.pending_count())
    # Library demo
    lib = Library()
    lib.add_book("978-0135166307", "Clean Code", "Robert C. Martin", copies=2)
    lib.add_book("978-045550206", "Fluent Python", "Luciano Ramalho", copies=1)
    lib.borrow_book("978-0135166307", "50001")
    lib.borrow_book("978-0135166307", "50001")
    lib.borrow_book("978-0135166307", "50001")
    print("Available books: ", lib.list_available())
    lib.return_book("978-0135166307", "50001")
    print("Available books after return: ", 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 B", now + timedelta(minutes=20))
    bs.add_route_time("Route B", now + timedelta(minutes=20))
    print("Next Route A: ", bs.get_next_stop("Route A", current_time=now))
    print("Routes in schedule: ", list(bs.route("Route A")))
    # Cafeteria orders demo
    caf = Cafeteria()
    caf.add_order("0001", "F001", "50001", ["Coffee", "Sandwich"], customer_type="Student")
    caf.add_order("0002", "F001", "50001", ["Salad"], customer_type="Faculty")
    caf.add_order("0003", "F002", "50002", ["Tea"], customer_type="Student")
    caf.add_order("0004", "F002", "50002", ["Pasta"], 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("Order demo complete.")

```

It is not giving any output.

I'll open + bug.py to verify its current contents and the `main` 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 `psql` command:

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

More Skip

Design data structures per feature (1/5)

1 file changed +103 -364

campus_resource_management.py

Describe what to build next

Ln 190 Col 29 Spaces: 4 UFT-8 CPU Python 3.11.0 (Microsoft Store) Go Live Preview

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.

