

Assignment 11.3 Ai Assisted Coding

Ht.no: 2303A510C0

Batch: 06

Task 1:

Smart Contact Manager (Arrays & Linked Lists)

Scenario

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

Prompt:

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

Code:

```
File Edit Selection < > AI Assistant Coding D -> File Explorer View Editor Search Terminal Help

# contact.py
# This file contains the logic for managing contacts.
# It uses a linked list to store contacts.

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

class ContactManager:
    def __init__(self):
        self.head = None
        self.current = None
        self.size = 0

    def add_contact(self, name, phone):
        new_contact = Contact(name, phone)
        if self.head is None:
            self.head = new_contact
            self.current = new_contact
        else:
            current = self.head
            while current.next is not None:
                current = current.next
            current.next = new_contact
            new_contact.prev = current
            self.current = new_contact
        print(f"Contact '{name}' added successfully!")
        self.size += 1

    def search_contact(self, name):
        current = self.head
        while current is not None:
            if current.name == name:
                return current
            current = current.next
        print(f"Contact '{name}' not found")
        return None

    def delete_contact(self, name):
        current = self.head
        while current is not None:
            if current.name == name:
                if current.prev is None:
                    self.head = current.next
                    if self.head is not None:
                        self.head.prev = None
                else:
                    current.prev.next = current.next
                    if current.next is not None:
                        current.next.prev = current.prev
                print(f"Contact '{name}' deleted successfully!")
                self.size -= 1
                return
            current = current.next
        print(f"Contact '{name}' not found")

    def list_contacts(self):
        current = self.head
        while current is not None:
            print(f"Name: {current.name}, Phone: {current.phone}")
            current = current.next

    def __str__(self):
        result = "List-based Contact Manager\n"
        result += "-----\n"
        result += f"Total contacts (loaded): {self.size}\n"
        result += "\n"
        return result

# main.py
# This file contains the main logic for the contact manager application.

if __name__ == "__main__":
    manager = ContactManager()
    manager.add_contact("John Doe", "123-4567")
    manager.add_contact("Jane Doe", "234-5678")
    manager.add_contact("Bob Smith", "345-6789")
    manager.add_contact("Alice Johnson", "456-7890")
    manager.list_contacts()

    contact = manager.search_contact("Jane Doe")
    if contact is not None:
        manager.delete_contact("Jane Doe")
        contact = manager.search_contact("Jane Doe")
        if contact is not None:
            print("Contact still exists after deletion")
        else:
            print("Contact deleted successfully")
    else:
        print("Contact not found")

    manager.list_contacts()
```

```
#!/usr/bin/python

# Manager class
class Manager:
    def __init__(self):
        self.contacts = []

    # Add contact
    def add_contact(self, name, email, phone):
        print("Adding contact: " + name)
        print("Email: " + email)
        print("Phone: " + phone)

        self.contacts.append((name, email, phone))

    # Get all contacts
    def list_contacts(self):
        print("List of contacts:")
        for contact in self.contacts:
            print(contact[0] + ", " + contact[1] + ", " + contact[2])

    # Search contact by name
    def search_contact(self, name):
        print("Searching for contact: " + name)
        results = []
        for contact in self.contacts:
            if contact[0] == name:
                results.append(contact)
        return results

    # Update contact
    def update_contact(self, old_name, new_name, email, phone):
        print("Updating contact: " + old_name)
        print("New name: " + new_name)
        print("Email: " + email)
        print("Phone: " + phone)

        for contact in self.contacts:
            if contact[0] == old_name:
                contact[0] = new_name
                contact[1] = email
                contact[2] = phone

    # Delete contact
    def delete_contact(self, name):
        print("Deleting contact: " + name)

        for contact in self.contacts:
            if contact[0] == name:
                self.contacts.remove(contact)

# Create Manager object
manager = Manager()

# Add contacts
manager.add_contact("John Doe", "johndoe@example.com", "555-1234")
manager.add_contact("Jane Doe", "janedoe@example.com", "555-1234")
manager.add_contact("Bob Smith", "bobsmith@example.com", "555-1234")
manager.add_contact("Alice Johnson", "alicejohnson@example.com", "555-1234")
manager.add_contact("David Wilson", "davidwilson@example.com", "555-1234")
manager.add_contact("Sarah Lee", "sarahlee@example.com", "555-1234")
manager.add_contact("Emily Davis", "emilydavis@example.com", "555-1234")
manager.add_contact("Michael Brown", "michaelbrown@example.com", "555-1234")
manager.add_contact("Laura Green", "lauragreen@example.com", "555-1234")
manager.add_contact("Kevin White", "kevinwhite@example.com", "555-1234")

# List contacts
print(manager.list_contacts())

# Search contact
print(manager.search_contact("John Doe"))

# Update contact
manager.update_contact("John Doe", "johndoe2@example.com", "555-12345")
print(manager.list_contacts())

# Delete contact
manager.delete_contact("John Doe")
print(manager.list_contacts())
print(manager.list_contacts())
```

Output:

```

File Edit Selection ...
PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL POSTMAN CONSOLE
diana@mail.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
c:\AI Assistant Coding
j& python contact_manager.py
Ran 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:
file.py
bug.py
Describe what to build next
1 Hidden T...
Ln 226 Col 1 Spaces: 4 UFT-8 Python 3.11.9 (Microsoft Store) ⓘ Go Live ⌂ Prettier

```

The terminal output shows the execution of 'bug.py' and its interaction with a linked list of contacts:

```

PS C:\AI Assistant Coding> & C:/Users/edula/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/AI Assistant Coding/bug.py"
--- All Contacts (Linked List) ---
ID: 4, Name: Diana Prince, Phone: 555-1004, Email: diana@mail.com
ID: 3, Name: Charlie Brown, Phone: 555-1003, Email: charlie@mail.com
ID: 2, Name: Bob Smith, Phone: 555-1002, Email: bob@mail.com
ID: 1, Name: Alice Johnson, Phone: 555-1001, Email: alice@mail.com

--- Search by ID ---
Found: ID: 4, Name: Diana Prince, Phone: 555-1004, Email: diana@mail.com

--- Search by Name ---
Search results for 'Alice': [ID: 1, Name: Alice Johnson, Phone: 555-1001, Email: alice@mail.com]

--- All Contacts (Linked List) ---
ID: 4, Name: Diana Prince, Phone: 555-1004, Email: diana@mail.com
ID: 3, Name: Charlie Brown, Phone: 555-1003, Email: charlie@mail.com
ID: 1, Name: Alice Johnson, Phone: 555-1001, Email: alice@mail.com

Total contacts (List): 3
--- All Contacts (Linked List) ---
ID: 4, Name: Diana Prince, Phone: 555-1004, Email: diana@mail.com
ID: 3, Name: Charlie Brown, Phone: 555-1003, Email: charlie@mail.com
ID: 1, Name: Alice Johnson, Phone: 555-1001, Email: alice@mail.com

Total contacts (List): 3
Total contacts (Linked List): 3
ID: 1, Name: Alice Johnson, Phone: 555-1001, Email: alice@mail.com

Total contacts (List): 3
Total contacts (Linked List): 3
ID: 3, Name: Charlie Brown, Phone: 555-1003, Email: charlie@mail.com

Total contacts (List): 3
Total contacts (Linked List): 3
ID: 1, Name: Alice Johnson, Phone: 555-1001, Email: alice@mail.com

Total contacts (List): 3
Total contacts (Linked List): 3
ID: 4, Name: Diana Prince, Phone: 555-1004, Email: diana@mail.com

Total contacts (List): 3
Total contacts (Linked List): 3
Total contacts (Linked List): 3
PS C:\AI Assistant Coding>

```

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 interface with the following details:

- File Explorer:** Shows files like `J.java`, `missy.py`, `evenoddsum.py`, `student.py`, `function.py`, `temperature.py`, `vote.py`, `missy.py`, `library_book_request.py`, and `contact_manager.py`.
- Editor:** The main editor pane displays Python code for a Priority Queue class and a demo program. The code includes methods for insertion, deletion, and processing requests from a queue.
- Bottom Status Bar:** Shows "Indexing completed.", "L 43 C 20", "Space 4", "UTF-8", "CHI", "Python", "1.1.0 [Microsoft Store]", "Go Live", and "Preview".
- Right Sidebar:** Features the "AI Assistant Coding" extension interface with sections for "Recent Requests", "Completed Programs", and "Run (path) command". It also includes a "Help" section with links to documentation and GitHub issues.

Output:

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

--- Priority Queue ---
1. ID: 3, Requester: Charlie (Student), Book: Web Development
2. ID: 5, Requester: Eve (Student), Book: Databases
Queue size: 2

=====
PS C:\AI Assistant Coding>
```

Explanation:

- Queue (FIFO) → First request comes, first served.(If a student requests first, they get the book first.)
 - Priority Queue → Faculty requests are served before students, even if they come later.
 - enqueue() → Adds a request to the system.
 - dequeue() → Removes and processes the next request.

Task 3: Emergency Help Desk (Stack Implementation)

Scenario

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

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

Prompt:

Write a program of stack-based ticket system in Python with push, pop, peek, isEmpty, and isFull. Add five tickets and show how they are resolved using LIFO order.

Code:

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

- File Explorer:** Shows a tree view of files under "ASSISTANT CODING".
- Editor:** Displays a Python script named `bug.py` containing code for a stack-based ticket system. The code includes methods for adding tickets to a stack (LIFO), removing them, checking if the stack is empty, and displaying all tickets.
- Output:** Shows the command "cd \"C:\AI Assistant Coding\" & python bug.py" and the response "[1] Resolving new ticket...".
- Terminal:** Shows the command "cd \"C:\AI Assistant Coding\" & python bug.py" and the response "[1] Resolving new ticket...".
- Status Bar:** Shows the file path "C:\AI Assistant Coding\bug.py" and other status indicators.

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

- File Explorer:** Shows a tree view of files in the current workspace, including Python scripts like `bug.py`, `library_book_request.py`, and `contact_manager.py`, as well as configuration files like `.gitignore`.
- Editor:** The main editor area displays the `bug.py` file. The code implements a stack-based ticket system. It includes functions for pushing tickets onto the stack, popping them off, and resolving all tickets in LIFO order.
- Terminal:** The bottom terminal window shows the command `PS C:\VAE Assistant Coding> cd "C:\Users\medya\Repos\betaalpha\local\Microsoft\KitematicApps\python3.11-exe"\& AI Assistant Coding\bug.py` being run.
- AI Assistant Coding:** A sidebar on the right is titled "RUNNING ASSISTANCE". It contains a message: "edit in this file without creating another file". Below this, there are two sections:
 - "Review and update log.py": Contains a note: "Now let me test the updated code".
 - "Write a program of stack-based ticket system in Python with push, pop, peek, isEmpty, and isFull. Add the tickets and show how they are resolved using LIFO order": A code editor with the following code:

```
# coding: utf-8

# Create ticket stack with max size of 10
ticket_system = TicketStack(max_size=10)

# Add five tickets
print("---- ADDING TICKETS TO THE STACK ----")
ticket_system.push(Ticket(101, "Jane Smith", "Login issue"))
ticket_system.push(Ticket(102, "John Doe", "Forgot password"))
ticket_system.push(Ticket(103, "Sarah Johnson", "Payment processing error"))
ticket_system.push(Ticket(104, "Mike Wilson", "Data export failed"))
ticket_system.push(Ticket(105, "Emma Brown", "Account suspended"))

# Display current stack
ticket_system.display_stack()

# Show stack information
print("---- STACK INFORMATION ----")
print(f"Stack Size: {ticket_system.size()}")
print(f"Is Stack Empty: {ticket_system.isEmpty()}")
print(f"Is Stack Full: {ticket_system.isFull()}")


# Peek at the top ticket
print("---- PEAK AT TOP TICKET ----")
top_ticket = ticket_system.peek()
if top_ticket:
    print(f"Top ticket (without removing): {top_ticket}")

# Resolve all tickets in LIFO order
ticket_system.resolve_all()

# Display final stack state
print("---- FINAL STACK STATE ----")
print(f"Stack Size: {ticket_system.size()}")
print(f"Is Stack Empty: {ticket_system.isEmpty()}")
ticket_system.display_stack()
```

Output:

Explanation:

The program uses a stack to manage help desk tickets.

A stack works in last in, first solved order.

When a new ticket is raised, it is added to the top.

When solving a ticket, the most recent one is handled first.

The program can also check if there are no tickets left or if the stack is full.

Task 4:

Hash Table

Objective

To implement a Hash Table and understand collision handling

Prompt:

Write a Python program to create a Hash Table.

Add methods to insert, search, and delete data.

Handle collisions using chaining (store multiple values in a list at the same index).

Add comments to explain the code and show example usage.

Code:

File Edit Selection ... ↶ 🔍 AI Assistant Coding

AI Assistant Coding

Work on Python program to create a Hash Table.
Add methods to insert, search, and delete data.
Handle collisions using chaining
(store multiple values at the same index).
Add comments to explain the code
and show example usage.

Updated log

Run gpt command

cd "C:\AI Assistant Coding" & python log.py

My

```
class HashTable:
    """
    A simple Hash Table with handling collisions.
    This class represents a key-value pair for storing data.
    (key, value) represents a key-value pair in the hash table
    Args:
        self.size (int): Number of buckets in the hash table
    """
    def __init__(self, size):
        self.size = size
        self.table = [None] * self.size
        self.total_items = 0

    def _hash(self, key):
        """
        Hash function to determine the index for a given key
        This simple modulo operation: hash(key) % table_size
        """
        return hash(key) % self.size

    def insert(self, key, value):
        """
        Insert a key-value pair into the hash table
        If key already exists, update its value
        If collision occurs, add to the linked list at that index
        """
        index = self._hash(key)
        if self.table[index] is None:
            self.table[index] = Node(key, value)
        else:
            self.table[index].insert(key, value)

        self.total_items += 1
        return True

    def search(self, key):
        """
        Search for a key in the hash table
        Args:
            key: The key to search for
        Returns:
            The value associated with the key, or None if not found
        """
        index = self._hash(key)
        if self.table[index] is None:
            return None
        else:
            return self.table[index].search(key)

    def delete(self, key):
        """
        Delete a key-value pair from the hash table
        Args:
            key: The key to delete
        Returns:
            bool: True if deletion successful, False if key not found
        """
        index = self._hash(key)
        if self.table[index] is None:
            return False
        else:
            self.table[index].delete(key)
            self.total_items -= 1
            return True

    def display(self):
        """
        Display all items in the hash table
        """
        for i, pair in enumerate(self.table):
            if pair is not None:
                print(f"Index {i}: {pair}")

    def __str__(self):
        return str(self.table)
```

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

[1] Resolving next ticket...
[2] Resolving ticket #30 | Customer: John Smith | Issue: Login issue

File Edit Selection ... ↶ 🔍 AI Assistant Coding

AI Assistant Coding

Work on Python program to create a Hash Table.
Add methods to insert, search, and delete data.
Handle collisions using chaining
(store multiple values at the same index).
Add comments to explain the code
and show example usage.

Updated log

Run gpt command

cd "C:\AI Assistant Coding" & python log.py

My

```
class HashTable:
    """
    A simple Hash Table with handling collisions.
    This class represents a key-value pair for storing data.
    (key, value) represents a key-value pair in the hash table
    Args:
        self.size (int): Number of buckets in the hash table
    """
    def __init__(self, size):
        self.size = size
        self.table = [None] * self.size
        self.total_items = 0

    def _hash(self, key):
        """
        Hash function to determine the index for a given key
        This simple modulo operation: hash(key) % table_size
        """
        return hash(key) % self.size

    def insert(self, key, value):
        """
        Insert a key-value pair into the hash table
        If key already exists, update its value
        If collision occurs, add to the linked list at that index
        """
        index = self._hash(key)
        if self.table[index] is None:
            self.table[index] = Node(key, value)
        else:
            self.table[index].insert(key, value)

        self.total_items += 1
        return True

    def search(self, key):
        """
        Search for a key in the hash table
        Args:
            key: The key to search for
        Returns:
            The value associated with the key, or None if not found
        """
        index = self._hash(key)
        if self.table[index] is None:
            return None
        else:
            return self.table[index].search(key)

    def delete(self, key):
        """
        Delete a key-value pair from the hash table
        Args:
            key: The key to delete
        Returns:
            bool: True if deletion successful, False if key not found
        """
        index = self._hash(key)
        if self.table[index] is None:
            return False
        else:
            self.table[index].delete(key)
            self.total_items -= 1
            return True

    def display(self):
        """
        Display all items in the hash table
        """
        for i, pair in enumerate(self.table):
            if pair is not None:
                print(f"Index {i}: {pair}")

    def __str__(self):
        return str(self.table)
```

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

[1] Resolving next ticket...
[2] Resolving ticket #30 | Customer: John Smith | Issue: Login issue

```

class HashTable:
    def __init__(self):
        self._size = 10
        self._table = [None] * self._size
        for pair in chain(*self._table):
            if pair is not None:
                self._table.append(pair)
        self._table.append(None)

    def get_all_items(self):
        """Return all key-value pairs in the hash table"""
        all_items = []
        for index in range(self._size):
            for pair in self._table[index]:
                if pair is not None:
                    all_items.append(pair)
        return all_items

    # ----- HIGH LEVEL PROGRAM -----
    def __str__(self):
        print("-----")
        print("----- HASH TABLE WITH CHAINING COLLISION RESOLVING -----")
        print("-----")
        print("----- Keys -----")
        for index in range(self._size):
            print(index)
        print("----- Values -----")
        for index in range(self._size):
            print(self._table[index])
        print("-----")

    # Create a hash table with 5 entries
    # Handle collisions using chaining
    hash_table = HashTable(5)

    # ----- INSERT OPERATIONS -----
    print("----- INSERTING DATA -----")
    hash_table.insert("age", 30)
    hash_table.insert("name", "Alice")
    hash_table.insert("email", "alice@email.com")
    hash_table.insert("phone", "123-1234")
    hash_table.insert("city", "New York") # This may collide with other keys
    hash_table.insert("salary", 75000)
    hash_table.insert("country", "USA")

    # Display the hash table
    hash_table.display()

    # ----- SEARCH OPERATIONS -----
    print("----- SEARCHING FOR DATA -----")
    hash_table.search("name")
    hash_table.search("age")
    hash_table.search("name") # Key that doesn't exist

    # ----- UPDATE OPERATIONS -----
    print("----- UPDATING DATA -----")
    hash_table.insert("age", 31) # Update existing key
    hash_table.delete("name") # Try to delete non-existent key
    hash_table.delete("name") # Try to delete non-existent key

    # Display the hash table after updates
    hash_table.display()

    # ----- DELETE OPERATIONS -----
    print("----- DELETING DATA -----")
    hash_table.delete("name") # Delete existing key
    hash_table.delete("name") # Try to delete non-existent key

    # Display the hash table after deletions
    hash_table.display()

    # ----- GET ALL ITEMS -----
    print("----- ALL REMAINING ITEMS -----")
    all_items = hash_table.get_all_items()
    for item in all_items:
        print(item)

    print(f"Total items in hash table: {len(all_items)}")

#----- HIGH TABLE CONTENTS -----
Index 0: (empty)
Index 1: name: Alice -> salary: 75000
Index 2: name: Alice -> city: New York -> department: IT
Index 3: name: Alice -> salary: 75000
Index 4: (empty)

Total items in hash table: 4

```

Output:

```

PS C:\AI Assistant Coding & C:\Users\mdua\PugData\Local\Microsoft\WindowsApps\python3.11.exe "C:\AI Assistant Coding\HashTable.py"

Index 0: (empty)
Index 1: name: Alice -> salary: 75000
Index 2: name: Alice -> city: New York -> department: IT
Index 3: name: Alice -> salary: 75000
Index 4: (empty)

Total items in hash table: 4

----- HIGH TABLE CONTENTS -----
Index 0: (empty)
Index 1: name: Alice -> salary: 75000
Index 2: name: Alice -> city: New York -> department: IT
Index 3: name: Alice -> salary: 75000
Index 4: (empty)

Total items in hash table: 4

```

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 ...

campus_resource_management.py

```

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

# ----- Library (dict/hash table) -----
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("Added copies (%d) more copies of (%s) (ISBN: %s)." % (copies, title, isbn))
        else:
            self.catalog[isbn] = {
                "title": title,
                "author": author,
                "copies": copies,
                "borrowers": []
            }
        print("Added book (%s) (ISBN: %s)." % (title, 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("%s borrowed (%s)." % (user_id, book["title"]))
        return True
    def return_book(self, isbn, user_id):
        book = self.catalog.get(isbn)
        if not book:
            print("Book not found.")
            return False
        try:
            book["borrowers"].remove(user_id)
        except ValueError:
            print("This user did not borrow the book.")
            return False
        book["copies"] += 1
        print("%s returned (%s)." % (user_id, book["title"]))
        return True
    def list_available(self):
        return [(isbn, info["title"], info["copies"]) for isbn, info in self.catalog.items()]
# ----- Bus Schedule (route -> sorted list of times) -----

```

File Edit Selection ...

campus_resource_management.py

```

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

# ----- Library -----
class Library:
    def __init__(self):
        self.attendance = {}
        self.attendance['Alice'] = 0
        self.attendance['Bob'] = 0
        self.attendance['Charlie'] = 0
    def present(self, name):
        if name in self.attendance:
            self.attendance[name] += 1
            print('Present count:', self.attendance[name])
        else:
            print('Absent student: %s' % name)
    def absent(self, name):
        if name in self.attendance:
            self.attendance[name] -= 1
            print('Present count:', self.attendance[name])
        else:
            print('Absent student: %s' % name)
    def process_registration(self):
        ev = EventRegistration()
        ev.add_registration('Alice')
        ev.add_registration('Bob')
        ev.add_registration('Charlie')
        print('Pending registrations:', ev.list_pending())
        ev.process_registration()
        print('Present count:', self.attendance['Alice'])
        print('Absent count:', self.attendance['Bob'])

# ----- Library -----
lib = Library()
lib.add_book("978-0151516307", "Clean Code", "Robert C. Martin", copies=2)
lib.add_book("978-0151516321", "Fluent Python", "Luciano Ramalho", copies=1)
lib.borrow_book("978-0151516307", "5000")
lib.borrow_book("978-0151516307", "5001")
lib.list_available()
print("Available books:", lib.list_available())
lib.return_book("978-0151516307", "5001")
print("Available books after return:", lib.list_available())

# ----- Bus schedule -----
bs = BusSchedule()
now = datetime.now()
bs.add_route_time("Route A", now + timedelta(minutes=5))
bs.add_route_time("Route A", now + timedelta(minutes=20))
bs.add_route_time("Route B", now + timedelta(minutes=2))
print("Next bus for Route A: %s ; in %d min ; Route A : current_time=%s" % (bs.next_bus(RouteA), bs.pending_time(RouteA), now))
print("Next bus for Route B: %s ; in %d min ; Route B : current_time=%s" % (bs.next_bus(RouteB), bs.pending_time(RouteB), now))

# ----- Cafeteria orders -----
caf = Cafeteria()
caf.add_order("0001", "0001", "Coffee", "Sandwich", customer_type="Student")
caf.add_order("0002", "F0001", "Salad", customer_type="Faculty")
caf.add_order("0003", "0002", "Tea", customer_type="Student")
caf.add_order("0004", "0003", "Pasta", customer_type="Faculty")
print("Pending cafeteria orders: ", caf.pending_count())
caf.serve_order()
caf.serve_order()
print("Running orders after serving: ", caf.pending_count())
print("\nDemo complete.\n")

```

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: O001 (Student)
Order added: O002 (Faculty)
Order added: O003 (Student)
Order added: O004 (Faculty)
Pending cafeteria orders: 4
Serving order: O002 (Faculty)
Serving order: O004 (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.