

ASSIGNMENT-11.3

Name: T.Swetha

Ht.no: 2303A51317

Batch: 05

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

Scenario

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

Prompt:

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

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

Code:

The screenshot shows a Python code editor with a file named `contact_manager.py` open. The code implements a contact manager using a linked list. It includes functions for adding contacts, searching by ID or name, displaying all contacts, deleting contacts, and printing statistics. The code is well-structured with comments explaining the logic.

```
#!/usr/bin/python

# A using linked Contact Manager
# python contact_manager.py
# manager_ll = ContactManager()

# Add contacts
# manager_ll.add_contact(contact_id="1", name="John Doe", "123 Main St", "johndoe@email.com")
# manager_ll.add_contact(contact_id="2", name="Jane Doe", "456 Elm St", "janedoe@email.com")
# manager_ll.add_contact(contact_id="3", name="Charlie Brown", "567 Oak St", "charliebrown@email.com")
# manager_ll.add_contact(contact_id="4", name="Snoopy", "789 Pine St", "snoopy@email.com")

# Display all
manager_ll.display_all()

# Search by ID
print("Search by ID ...")
contact = manager_ll.search_contact(2)
if contact:
    print(f"Found: {contact}")

# Search by Name
print("Search by Name ...")
print(manager_ll.search_name("Doe"))

# Delete contact
print("Delete contact ...")
manager_ll.delete_contact(3)
manager_ll.display_all()

# Search using linked list ...
manager_ll + ContactManager().__str__()

# Example usage
# manager_ll.add_contact(contact_id="1", name="John Doe", "123 Main St", "johndoe@email.com")
# manager_ll.add_contact(contact_id="2", name="Jane Doe", "456 Elm St", "janedoe@email.com")
# manager_ll.add_contact(contact_id="3", name="Charlie Brown", "567 Oak St", "charliebrown@email.com")
# manager_ll.add_contact(contact_id="4", name="Snoopy", "789 Pine St", "snoopy@email.com")

# Print stats
print("Total contacts (linked list):", len(manager_ll))
print("Total contacts (linked list):", len(manager_ll[1]))
```

Output:

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

- Title Bar:** File Edit Selection ... diana@email.com No matching results
- Explorer View (Left):** Shows a file tree with various files including J fib.java, assv.py, evenmodsum.py, student.py, function.py, TEMPERATURE.PY, vote.py, buggy.py, and contact_manager.py.
- Terminal View (Bottom):** Displays the output of a Python script named "buggy.py". The script performs several search operations on a list of contacts (Diana Prince, Charlie Brown, Bob Smith, Alice Johnson) and prints the total contact counts for each search query.
- Right Side Panel:** BUG FIXING ASSISTANT... (disabled), Write a python program to Create a Contact Manager using list and linked list with add, search and delete functions, Planned Contact Manager (with a checklist), I'll create a comprehensive Contact Manager program with both list and linked list implementations, Created contact_manager.py, and Run contact manager (with a checklist).

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

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

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

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

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

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

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

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

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

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.

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:** Q. AI Assistant Coding
- File Explorer (Left):** Shows a tree view of files and folders, including Java, Python, and XML files.
- Code Editor (Main):** Displays Python code for a library management system. The code defines two classes: `NormalQueue` and `PriorityBookRequest`.
 - `NormalQueue` handles requests for books. It has methods for adding requests to a queue, removing the first request, and displaying the queue. It also includes a `size` method and a `__str__` method for printing the queue state.
 - `PriorityBookRequest` handles requests with priority (Faculty > Student). It includes methods for getting the priority level (lower number = higher priority) and comparing user types.
- Output (Bottom Right):** Shows logs related to the code execution, including messages from the AI Assistant and build status.
- Status Bar (Bottom):** Shows the current file path (bug.py), line number (Line 26), column (Col 50), and other standard status indicators.

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

- File Explorer:** Shows the project structure with files like `bug.py`, `library_book_request.py`, and `contact_manager.py`.
- Open Editors:** Displays multiple tabs for Python files:
 - `bug.py`: Contains a `PriorityQueue` class and a `remove_request` function.
 - `library_book_request.py`: Contains a `NormalQueue` class and a `remove_request` function.
 - `contact_manager.py`: Contains a `NormalQueue` class and a `remove_request` function.
- AI Assistant Coding:** A sidebar feature showing AI-generated code snippets for various tasks like `BUS FIXING...`, `AI Assistant Coding`, and `BUG FIXING...`.
- Code Snippets:** A right-hand panel showing code examples for `BUS FIXING...`, `AI Assistant Coding`, and `BUG FIXING...`.
- Status Bar:** Shows the current file path as `C:\Users\Aman\PycharmProjects\Assignment-3\bug.py` and the line number as `L 119`.

```

File Edit Selection ...
File Explorer
OPEN EDITORS
J虫Java aspy evenoddum.py student.py function.py temperaturePY vote.py library_book_request.py contact_manager.py
AI ASSISTANT CODING
J虫Java aspy evenoddum.py student.py function.py temperaturePY vote.py library_book_request.py contact_manager.py
BUG FIXING ...
requests are given first priority over student requests. include functions to add a request and remove a request.
✓ Create file
Executed program to demonstrate book system working
6) Run push command!
cd "c:\AI Assistant Coding"
python bug.py
ok_request
OK
Now let me test the updated code
7) Run push command!
cd "c:\AI Assistant Coding"
python bug.py
bug.py
OK
Edit in this file without creation of another file
✓ Reviewed and updated bug.py
Now let me test the updated code
8) Run push command!
cd "c:\AI Assistant Coding"
python bug.py
bug.py
OK
Go Live
Describe what to build
Q < > < > < > < >
Ln 41 Col 20 Spaces: 4 UFT-8 CRLF Python 3.11.9 (Microsoft Store)

```

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.

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

Code:

File Edit Selection ...

AI Assistant Coding

REVIEWED AND UPDATED bug.py

Now let me test the updated code:

```

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

```

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

Reviewed and updated bug.py and checked additional lines

Now let me test the stack-based ticket system:

```

Run [shell] command!
cd "c:\AI Assistant Coding" & python bug.py
Allow Skip

```

PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL POSTMAN CONSOLE

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

[3] Resolving next ticket...

File Edit Selection ...

AI Assistant Coding

REVIEWED AND UPDATED bug.py

Now let me test the updated code:

```

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

```

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

Reviewed and updated bug.py and checked additional lines

Now let me test the stack-based ticket system:

```

Run [shell] command!
cd "c:\AI Assistant Coding" & python bug.py
Allow Skip

```

PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL POSTMAN CONSOLE

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

[3] Resolving next ticket...

Output:

The screenshot shows a Microsoft Visual Studio Code interface. The Explorer sidebar on the left lists several Python files: bug.py, everodump.py, function.py, and library_book_request.py, along with some JSON and XML files. The Terminal tab at the top shows a command-line session where the user runs a script named 'bug.py'. The output of this script is displayed in the terminal, showing a stack-based ticket system. The AI Assistant Coding feature is active, with a sidebar on the right showing generated code for ticket resolution and instructions for testing the system.

Explanation:

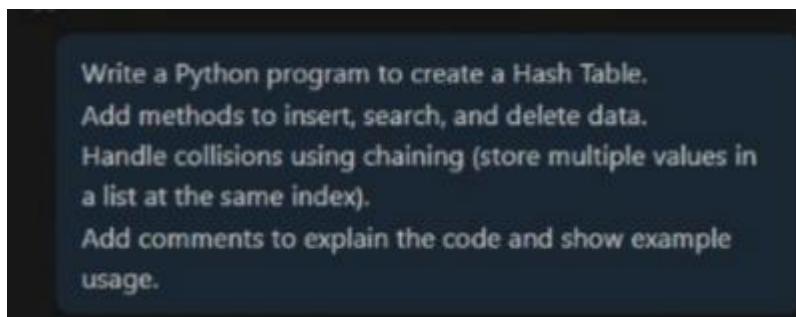
The program uses a stack to manage help desk tickets. A stack works in last in, first solved order. When a new ticket is raised, it is added to the top. When solving a ticket, the most recent one is handled first. The program can also check if there are no tickets left or if the stack is full.

Task 4: Hash Table Objective

To implement a Hash Table and understand collision handling.

Prompt:

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



Code:

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

- File Structure:** The left sidebar shows a tree view of files and folders, including Java, Python, and C/C++ files, as well as various assignment and test files.
- Code Editor:** The main editor area contains Python code for a Hash Table with Chaining Collision Handling. The code includes a HashTable class with __init__ and __str__ methods, and a hash function. The code is annotated with comments explaining its functionality.
- Right Panel:** A "BUG FIXING ASSISTANCE" panel is open on the right, showing code snippets and search functions related to bug fixing.
- Status Bar:** The bottom status bar displays "PS | AI Assistant Coding: & C:\Users\mila\Applia\local\Microsoft\AI\hash_table\python\11.ee" and "Python 3.9 Microsoft Store".

File Edit Selection ...

AI Assistant Coding

```

J\Java easy evendiscovery studentpy function.py temperaturePY votepy buggy library_book_requestpy contact_managerpy
```

buggy >

```

class HashTable:
    def insert(self, key, value):
        # Check if key already exists in the chain and update if found
        for pair in self.chains:
            if pair.key == key:
                pair.value = value
                print(f"Value updated: {key} = {value} (at index {index})")
                return True

        # If key not found, add new key-value pair to the chain
        chain.append(KeyValuePair(key, value))

    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
        ...
        # Get the hash index
        index = self._hash_function(key)
        chain = self.chains[index]

        # Search for the key in the chain
        for pair in chain:
            if pair.key == key:
                print(f"Value found: {key} = {pair.value} (at index {index})")
                return pair.value

        # Key not found
        print(f"X Key '{key}' not found in the hash table")
        return None

    def delete(self, key):
        ...
        Delete a key-value pair from the hash table
        Args:
            key: The key to delete
        Returns:
            Boolean True if deletion successful, False if key not found
        ...
        # Get the hash index
        index = self._hash_function(key)
        chain = self.chains[index]

        # Search for and remove the key from the chain
        for i, pair in enumerate(chain):
            if pair.key == key:
                deleted_pair = chain.pop(i)
                self.total_items -= 1
                print(f"Deleted pair: {key} = {deleted_pair.value} (from index {index})")
                return True

        # Key not found
        print(f"X Key '{key}' not found in the hash table")
        return False

    def display(self):
        ...

```

PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL POSTMAN CONSOLE

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

[3] Resolving next ticket...
✓ Resolving Ticket #30 | Customer: John Smith | Issue: Login issue

File Edit Selection ...

AI Assistant Coding

```

J\Java easy evendiscovery studentpy function.py temperaturePY votepy buggy library_book_requestpy contact_managerpy
```

buggy >

```

class HashTable:
    def get_all_items(self):
        """Get all items stored in the hash table"""
        all_items = []
        for chain in self.chains:
            for pair in chain:
                all_items.append(pair)
        return all_items

    # ----- DEMO PROGRAM -----
    if __name__ == "__main__":
        print("HASH TABLE WITH CHAINING COLLISION HANDLING")
        print("-----")
        # Create a hash table with 5 buckets
        hash_table = HashTable(5)

        # Insert some data
        hash_table.insert("name", "Alice")
        hash_table.insert("name", "Bob")
        hash_table.insert("city", "New York")
        hash_table.insert("city", "Los Angeles")
        hash_table.insert("phone", "555-1234")
        hash_table.insert("country", "USA") # This may collide with other keys
        hash_table.insert("department", "IT")
        hash_table.insert("department", "IT")

        # Display the hash table
        hash_table.display()

        # ----- SEARCH OPERATIONS -----
        print("----- SEARCHING FOR DATA -----")
        hash_table.search("name", "Alice")
        hash_table.search("city", "New York")
        hash_table.search("phone", "555-1234")
        hash_table.search("country", "USA") # Key that doesn't exist
        hash_table.search("unknown_key") # Key that doesn't exist

        # ----- UPDATE OPERATIONS -----
        print("----- UPDATING DATA -----")
        hash_table.insert("age", 33) # Update existing key
        hash_table.display()

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

        # ----- DELETE OPERATIONS -----
        print("----- DELETING DATA -----")
        hash_table.delete("name", "Alice")
        hash_table.delete("city", "New York")
        hash_table.delete("phone", "555-1234")
        hash_table.delete("country") # Try to delete non-existent key
        hash_table.display() # Display the hash table after deletion

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

        print(f"Total items: {hash_table.get_size()}")
        print(f"Is empty: {hash_table.is_empty()}")

```

PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL POSTMAN CONSOLE

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

[3] Resolving next ticket...
✓ Resolving Ticket #30 | Customer: John Smith | Issue: Login issue

Output:

```

PS C:\AI Assistant Coding> & C:/Users/edua/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/AI Assistant Coding/bug.py"
Total Items in hash table: 8
--- SEARCHING FOR DATA ---
Found name = Alice (at index 2)
Found age = 30 (at index 1)
X Key nonexistent not found in the hash table
Total Items in hash table: 8
--- UPDATING DATA ---
Updated: age = 31 (at index 1)
Total Items in hash table: 8
--- DELETING DATA ---
Deleted: email = alice@mail.com (from index 0)
Deleted: city = New York (from index 1)
X Key "nonexistent" not found in the hash table
Total Items in hash table: 6
--- ALL REMAINING ITEMS ---
age: 31
city: New York
name: Alice
salary: 75000
phone: 555-1234
country: USA
Total Items: 6
Is empty: False

```

Explanation:

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

Task 5: Real-Time Application Challenge

Scenario

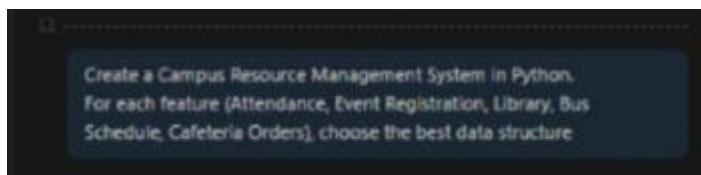
Design a Campus Resource Management System with the following features:

- Student Attendance Tracking

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

Prompt:

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



Code:

```

File Edit Selection ... < > AI Assistant Coding
EXPLORER
OPEN EDITORS
  campus_resource_management.py
  assopy
  evenodsum.py
  student.py
  function.py
  TEMPERATUREPY
  votey
  campus_resource_management.py
  bugpy (B)
  library_book_request.py
  contact_m
  ...
CMF
BUG FIXING ASSISTANCE
It is not giving any output.
I'll open + bugpy to verify its current contents and the _main_ section.
✓ Reviewed bugpy and fixed file structure.
I'll fix + bugpy by replacing it with a standalone Campus Resource Management System so the demo runs at module level.
Deleted bugpy.
Created + bugpy.
Run py main command!
cd "c:\AI Assistant Coding" ; python bugpy
More Skip
campus_resource_management.py
# Features and Data Structures:
# 1. Student ID set (O(1)) add/check to track present student IDs
# 2. Library: dict (hash table) mapping ISBN -> book record for fast lookup
# 3. Bus Schedule: dict of route -> sorted list of departure times (list kept sorted)
# 4. Cafeteria Orders: heapq (priority queue) to prioritize faculty over students while preserving arrival order
# Run this file to see a small demo of each feature.

from collections import deque
import heapq
import bisect
from bisect import insort
from datetime import timedelta, timezone
# ..... Attendance (set) .....
class Attendance:
    """Track attendance using a set for O(1) add/remove/check."""
    def __init__(self):
        self.present = set()
    def mark_present(self, student_id):
        self.present.add(student_id)
        print(f"Marked present: {student_id}")
    def mark_absent(self, student_id):
        self.present.discard(student_id)
        print(f"Marked absent: {student_id}")
    def is_present(self, student_id):
        return student_id in self.present
    def present_count(self):
        return len(self.present)
# ..... Event Registration (FIFO queue) .....
class EventRegistration:
    """Register attendees in arrival order using deque."""
    def __init__(self):
        self.queue = deque()
    def add_registration(self, attendee_id, name):
        self.queue.append((attendee_id, name))
        print(f"Registered: {attendee_id} - {name}")
    def process_registration(self):
        if not self.queue:
            print("No registrations to process.")
            return None
        attendee = self.queue.popleft()
        print(f"Processing registration: [attendee[{0}]] - [{attendee[1]}]")
        return attendee
    def pending_count(self):
        return len(self.queue)
# ..... Library Book Borrowing .....
class LibraryBookRequest:
    def __init__(self):
        self.books = {}
    def borrow_book(self, isbn):
        if isbn in self.books:
            self.books.pop(isbn)
            print(f"Book borrowed: {isbn}")
        else:
            print(f"Book not found: {isbn}")
    def return_book(self, isbn):
        self.books[isbn] = 1
        print(f"Book returned: {isbn}")
    def check_availability(self, isbn):
        if isbn in self.books:
            print(f"Book available: {isbn}")
        else:
            print(f"Book not available: {isbn}")
# ..... Bus Schedule .....
class BusSchedule:
    def __init__(self):
        self.routes = {}
    def add_route(self, route_id, departures):
        self.routes[route_id] = departures
    def get_departures(self, route_id):
        return self.routes.get(route_id, [])
# ..... Cafeteria Orders .....
class CafeteriaOrders:
    def __init__(self):
        self.orders = []
    def place_order(self, faculty_id, item):
        heapq.heappush(self.orders, (faculty_id, item))
    def get_orders(self):
        return self.orders
# ..... Temperature Monitoring .....
class TemperatureMonitoring:
    def __init__(self):
        self.temperatures = {}
    def add_temperature(self, location, temperature):
        self.temperatures[location] = temperature
    def get_temperature(self, location):
        return self.temperatures.get(location, None)
# ..... Voting System .....
class VotingSystem:
    def __init__(self):
        self.votes = {}
    def cast_vote(self, voter_id, choice):
        self.votes[voter_id] = choice
    def get_results(self):
        results = {}
        for voter_id, choice in self.votes.items():
            if choice in results:
                results[choice] += 1
            else:
                results[choice] = 1
        return results
# ..... Assigment Management .....
class AssignmentManagement:
    def __init__(self):
        self.assignments = {}
    def add_assignment(self, assignment_id, due_date):
        self.assignments[assignment_id] = due_date
    def get_assignments(self):
        return self.assignments
# ..... Student Management .....
class StudentManagement:
    def __init__(self):
        self.students = {}
    def add_student(self, student_id, name):
        self.students[student_id] = name
    def get_students(self):
        return self.students
# ..... Functionality .....
def main():
    # Implement the logic for each feature here based on the chosen data structures.
    # For example, for Event Registration, you might use a deque to store registrations
    # and then process them sequentially. For Library Book Borrowing, you might use a
    # dictionary to map ISBNs to books.
    pass

if __name__ == "__main__":
    main()

```

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

- File Explorer:** Shows multiple Python files in the workspace, including `ass.py`, `evereodsum.py`, `student.py`, `functionary`, `temperature.py`, `volt.py`, `campus_resource_management.py`, `bug.py`, `library_book_request.py`, and `contact_m`.
- Editor:** The main editor pane displays the `campus_resource_management.py` file, which contains code for managing library resources.
- AI Assistant Coding:** A sidebar titled "AI Assistant Coding" is open, showing the following text:

```
I'm not getting any output
I'll open a bugpy to verify its current contents and the _walk section.
✓ Reviewed bugpy and fixed file structure
I'll fix a bug by replacing it with a standalone Campus Resource Management System so the demo runs at model level.
    ⚡ Delete bugpy
    ✎ Create → bugpy
    ⚡ Run `walk` command!
cd "C:\IAAT Assistant Coding" & python bug.py
Allow Stop
```
- Bottom Status Bar:** Shows "Le 190, Col 20" and "Spaces: 4 UFT-8 CR LF () Python 1.19 Microsoft Store Go Live Preview".

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

- File Explorer:** Shows the project structure with files like `ass.py`, `evenodsum.py`, `student.py`, `function.py`, `temperature.py`, `vote.py`, `campus_resource_management.py`, `bug.py`, `library_book_request.py`, and `contact_m.py`.
- Editor:** The main editor area displays a Python script for a "Campus Resource Management System". The code includes functions for managing attendance, event registrations, library books, and cafeteria orders.
- Right Panel:** An "AI Assistant Coding" panel is open, showing the output of the AI's analysis:
 - It states "It is not giving any output".
 - A note says "I'll open + bug.py to verify its current contents and the __init__ section."
 - Review status: "Reviewed bug.py and fixed file structure".
 - A fix suggestion: "I'll fix + bug.py by replacing it with a standalone Campus Resource Management System so the demo runs at module level."
 - Buttons for "Delete bug.py", "Create", and "Run peek command!".
 - A terminal window shows the command: "cd "/c/VAI Assistant Coding"; python bug.py".
- Bottom Status Bar:** Shows "In 190 Col 29 Status A UFT-B CRU" and "1118 Microsoft Store Go Live".

Output:

```
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: 978-0135166307).
Added book: Fluent Python (ISBN: 978-1491958296).
S001 borrowed Clean Code
S003 borrowed Clean Code
No copies available.
Available books: [('978-0135166307', 'Clean Code', 0), ('978-1491958296', 'Fluent Python', 1)]
S001 returned Clean Code
Available books after return: [('978-0135166307', 'Clean Code', 1), ('978-1491958296', 'Fluent Python', 1)]
Added bus time for Route A: 2026-02-18 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.

