**Task Description #1 – Stack Implementation**

**Task: Use AI to generate a Stack class with push, pop, peek, and is\_empty methods.**

**Sample Input Code:**

**class Stack:**

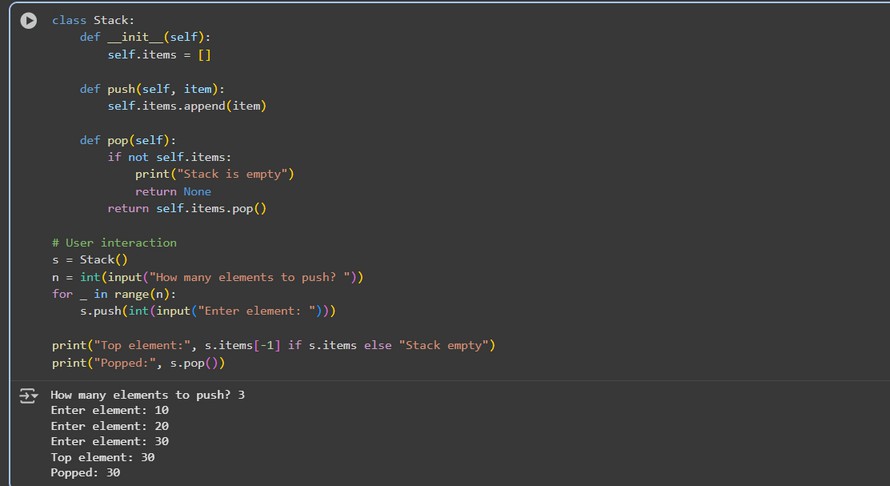
**pass**

**Expected Output:**

**A functional stack implementation with all required methods and docstrings.**

prompts :.

Create a Python class named Stack with methods: push, pop, peek, and is\_empty. Include docstrings and make it interactive to take user inputs for push operations



observations and code expalnation

 Defined Stack class with a list to store elements.  push(item) → adds an element to the top of stack.

 pop() → removes and returns the top element; handles empty stack.  peek() → shows top element without removing it.

 is\_empty() → checks if stack is empty.

 Interactive code asks for number of elements → pushes elements from user input.

Task Description #2 – Queue Implementation

Task: Use AI to implement a Queue using Python lists.

Sample Input Code:

class Queue:

pass

Expected Output:

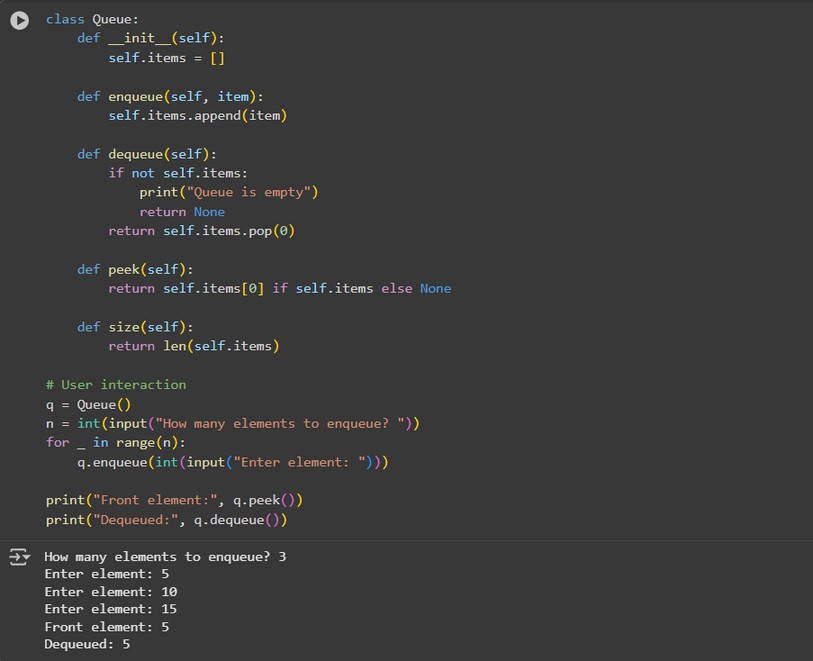
• FIFO-based queue class with enqueue, dequeue, peek, and size methods.

prompts :

Create a Python class named Queue using list. Include methods: enqueue, dequeue,

peek, and size. Make it interactive for user input.

code:



observations and code expalnation :

 Queue class stores elements in a list.

 enqueue(item) → adds element at the rear.

 dequeue() → removes element from front; prints message if empty.  peek() → shows front element.

 size() → returns number of elements.

 User input used to enqueue multiple elements; then displays front and dequeued element.

**Task Description #3 – Linked List**

Task: Use AI to generate a Singly Linked List with insert and display methods.

Sample Input Code:

class Node:

pass

class LinkedList:

pass

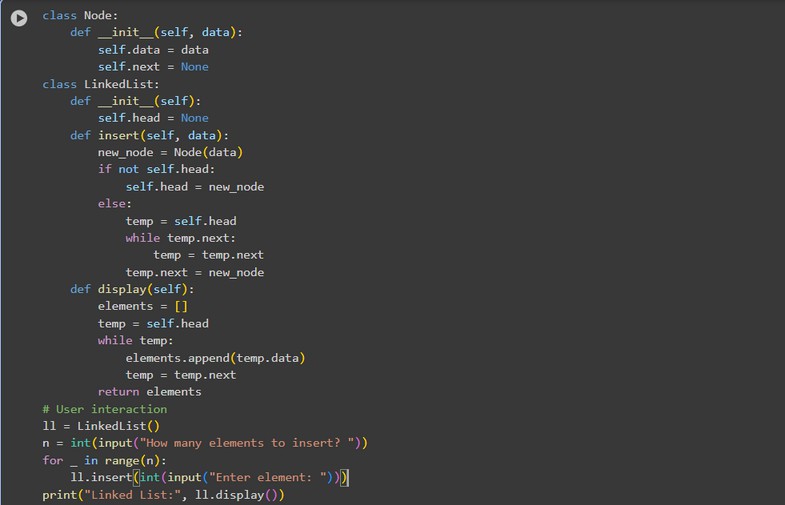
Expected Output:

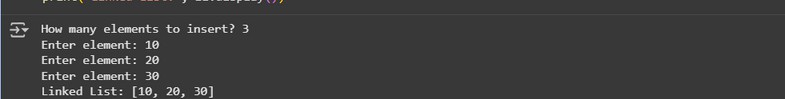
A working linked list implementation with clear method documentation

Prompts :

Create a Python Singly Linked List with Node class. Include insert and display methods. Make it interactive for user input.

Code:





Code explanation and observations :

 Node class stores data and next pointer.  LinkedList class has head pointer.

 insert(data) → adds new node at end.  display() → prints all nodes in order.

 User inputs number of nodes → program inserts each one and displays list.

**Task Description #4 – Binary Search Tree (BST)**

Task: Use AI to create a BST with insert and in-order traversal methods.

Sample Input Code:

class BST:

pass

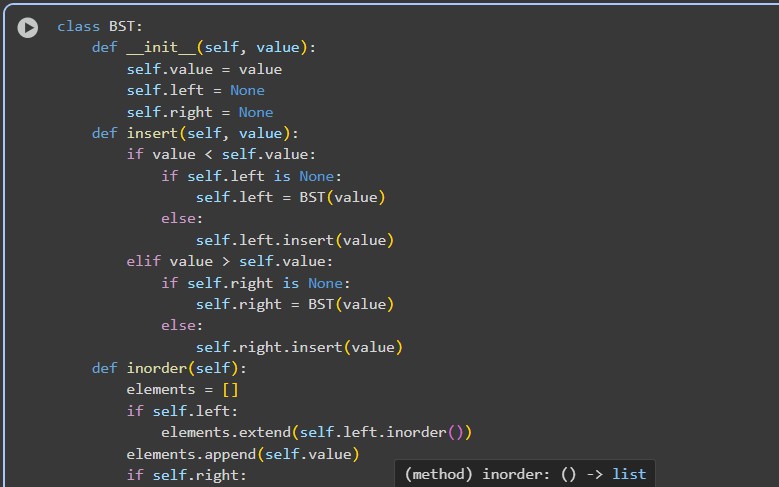
Expected Output:

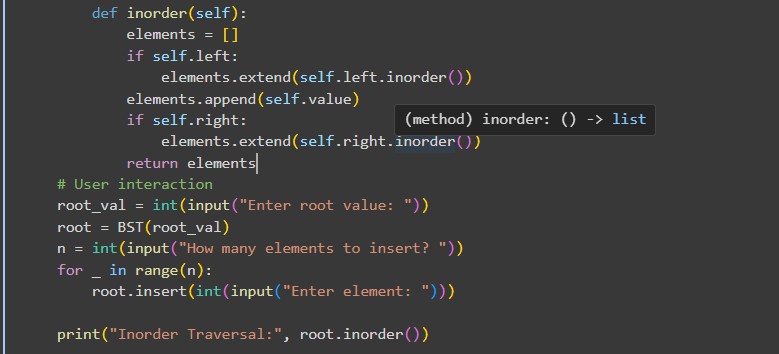
BST implementation with recursive insert and traversal methods

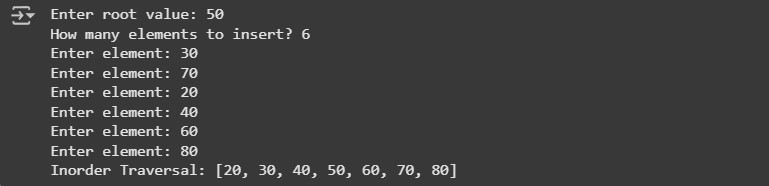
Prompts :

Create a Python BST class with insert and inorder traversal methods. Make it interactive for user input.

Code :







observations and code expalnation

BST node has data, left, right.

insert(data) → recursively places node in correct position. inorder() → returns elements in sorted order.

Program asks for root, then number of elements → inserts each. Prints inorder traversal to verify BST structure.

**Task Description #5 – Hash Table**

Task: Use AI to implement a hash table with basic insert, search, and delete methods.

Sample Input Code:

class HashTable:

pass

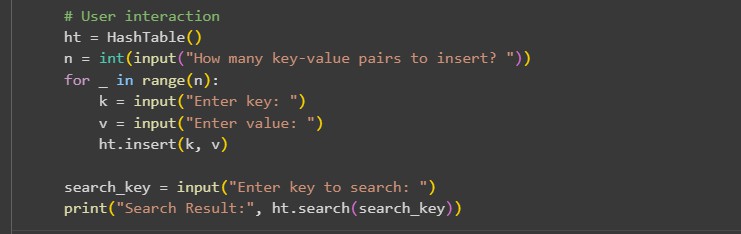
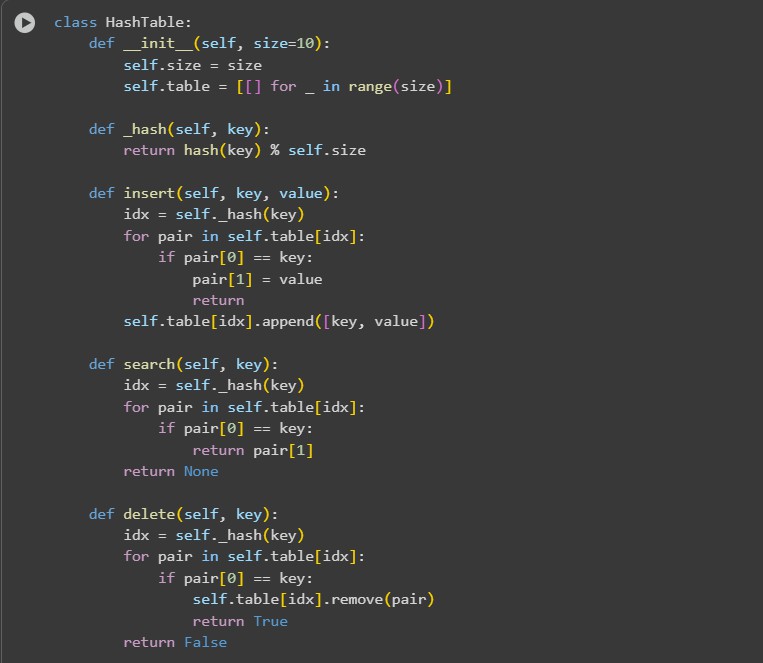
Expected Output:

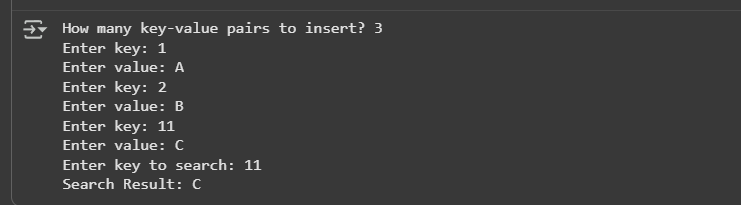
Collision handling using chaining, with well-commented methods

Prompts :

Create a Python Hash Table class with insert, search, and delete methods. Handle collisions using chaining. Include docstrings and user input.

Code:





Observation and code explanation

Uses a list of lists (buckets) for collision chaining. insert(key, value) → hashes key → appends to bucket.

search(key) → looks for key in bucket → returns value or None. delete(key) → removes key-value pair if exists.

User enters number of key-value pairs → program inserts → searches and prints result.

**Task Description #6 – Graph Representation**

Task: Use AI to implement a graph using an adjacency list.

Sample Input Code:

class Graph:

pass

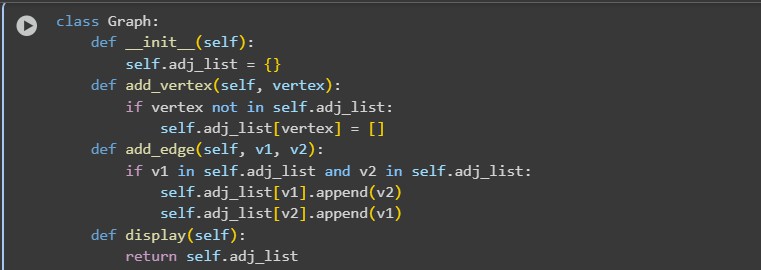
Expected Output:

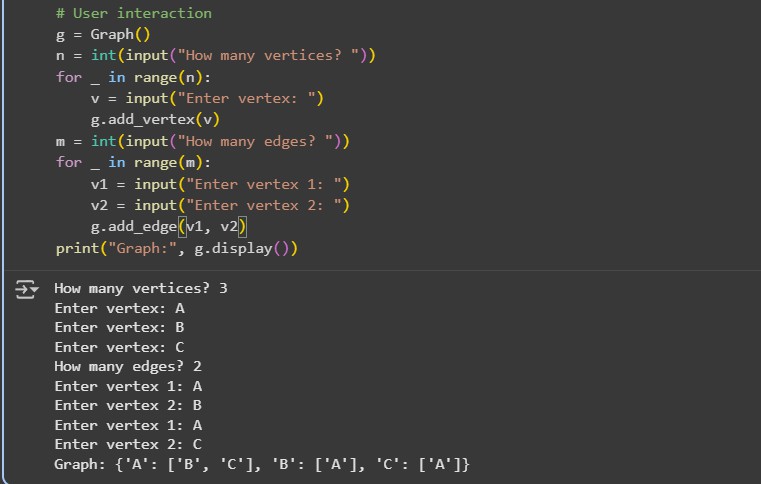
Graph with methods to add vertices, add edges, and display connections

Prompt :

Create a Python Graph class using adjacency list. Include methods: add\_vertex, add\_edge, display. Make it interactive.

Code





Code explanation and Observation :

 Dictionary stores adjacency list.

 add\_vertex(v) → adds vertex if not exist.

 add\_edge(v1, v2) → adds edge (undirected) between two vertices.  display() → prints adjacency list.

 Program asks number of vertices and edges → user enters each → displays graph.

**Task Description #7 – Priority Queue**

Task: Use AI to implement a priority queue using Python’s heapq module.

Sample Input Code:

class PriorityQueue:

pass

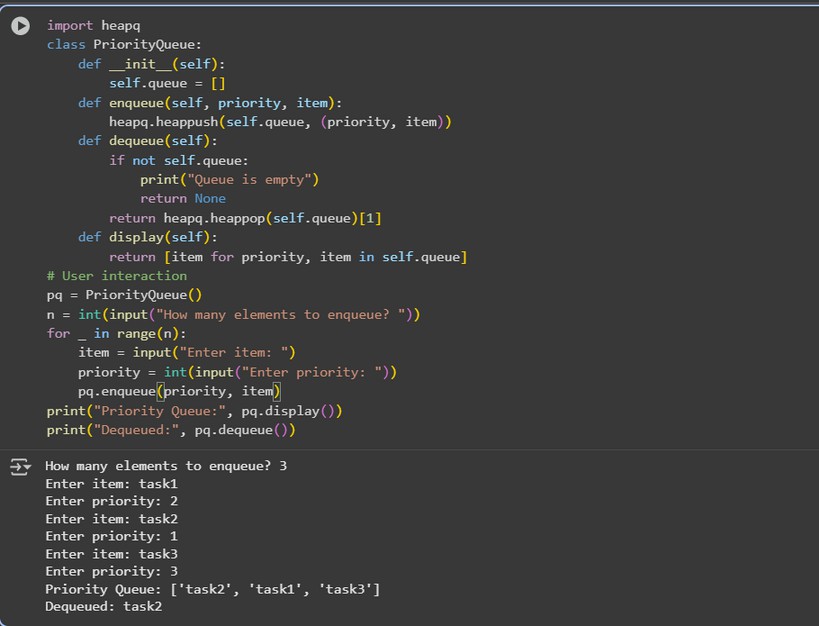
Expected Output:

Implementation with enqueue (priority), dequeue (highest priority), and display methods.

Prompt :

Create a Python Priority Queue class using heapq. Include enqueue (priority), dequeue (highest priority), and display methods. Interactive user input.

Code :



Observation and code explanation:

Uses heapq for priority management. enqueue(priority, item) → pushes tuple (priority, item). dequeue() → pops element with smallest priority value. display() → shows queue.

User enters items with priority → program enqueues → dequeues highest priority → displays queue.

**Task Description #8 – Deque**

Task: Use AI to implement a double-ended queue using collections.deque.

Sample Input Code:

class DequeDS:

pass

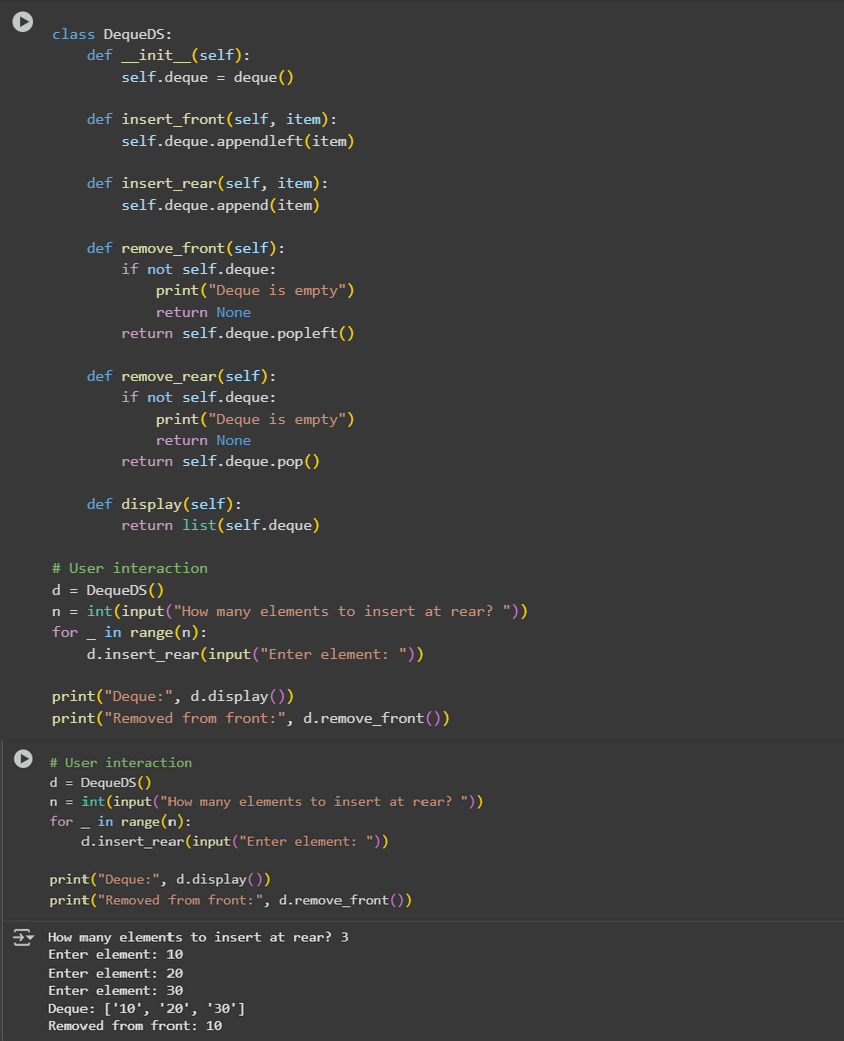
Expected Output:

Insert and remove from both ends with docstrings

Prompt :

Create a Python Deque class using collections.deque. Include methods: insert\_front, insert\_rear, remove\_front, remove\_rear, display. Interactive input.

Code :



Observation and code explation

 collections.deque allows fast insertion/removal at both ends.  insert\_front(item) → adds to front.

 insert\_rear(item) → adds to rear.

 remove\_front() / remove\_rear() → removes from respective end.  display() → prints deque.

 User inputs number of elements → inserts → removes → displays deque.

**Task Description #9 – AI-Generated Data Structure Comparisons**

Task: Use AI to generate a comparison table of different data structures (stack, queue, linked list, etc.) including time

complexities.

Sample Input Code:

# No code, prompt AI for a data structure comparison table

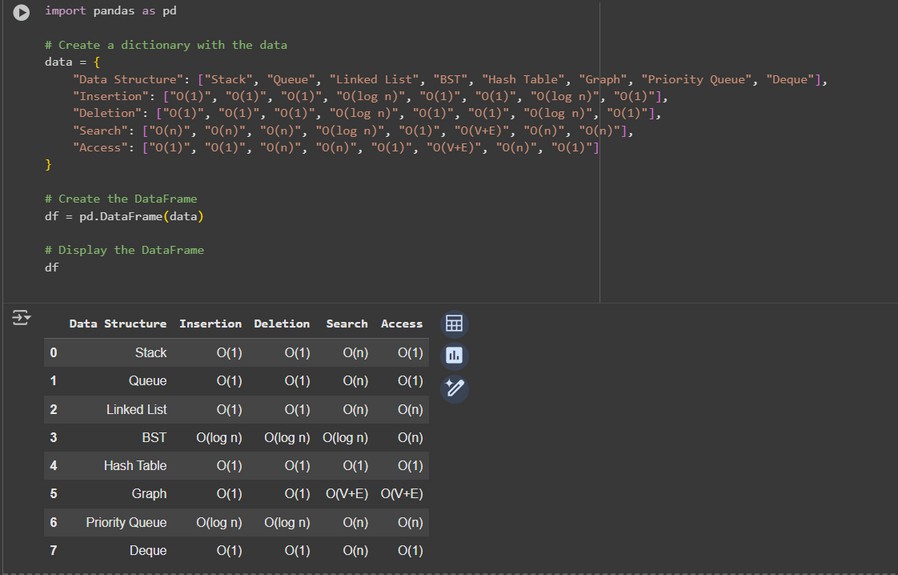
Expected Output:

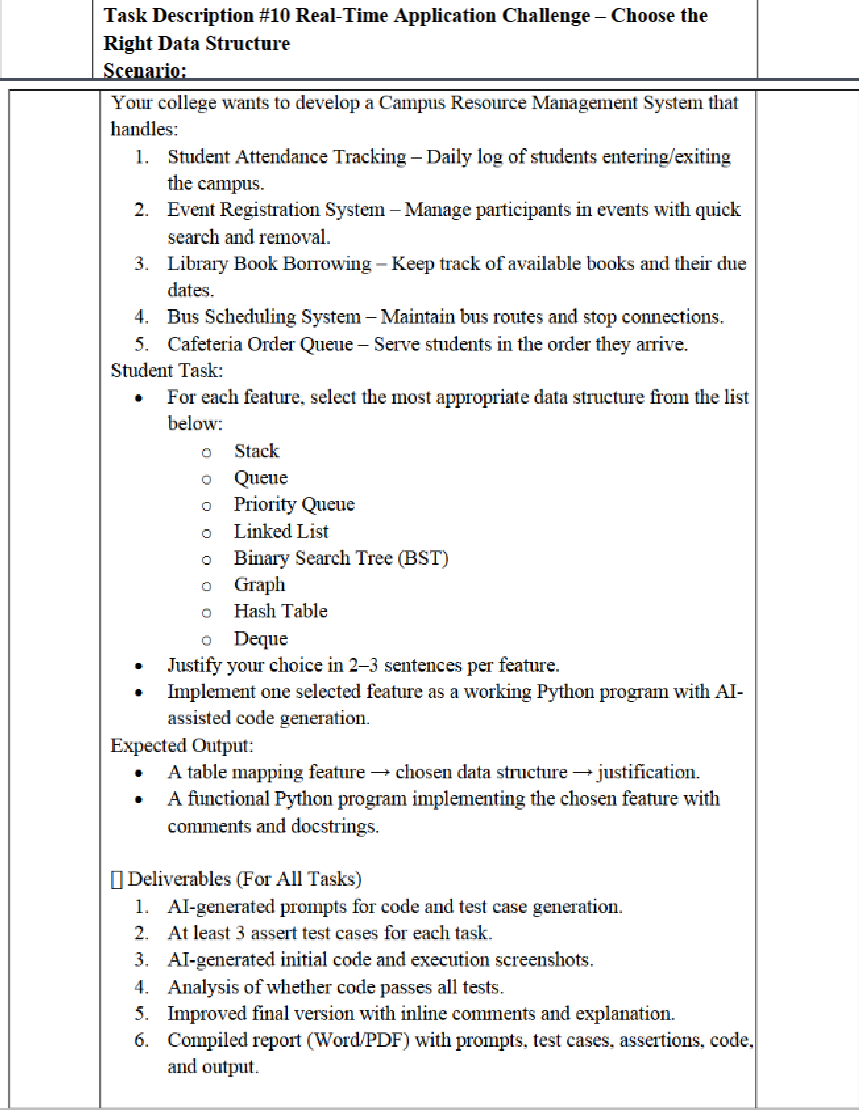
A markdown table with structure names, operations, and complexities

Prompt :

Generate a markdown table comparing Stack, Queue, Linked List, BST, Hash Table, Graph, Priority Queue, Deque with their time complexities for insertion, deletion, search, access. Using pandas

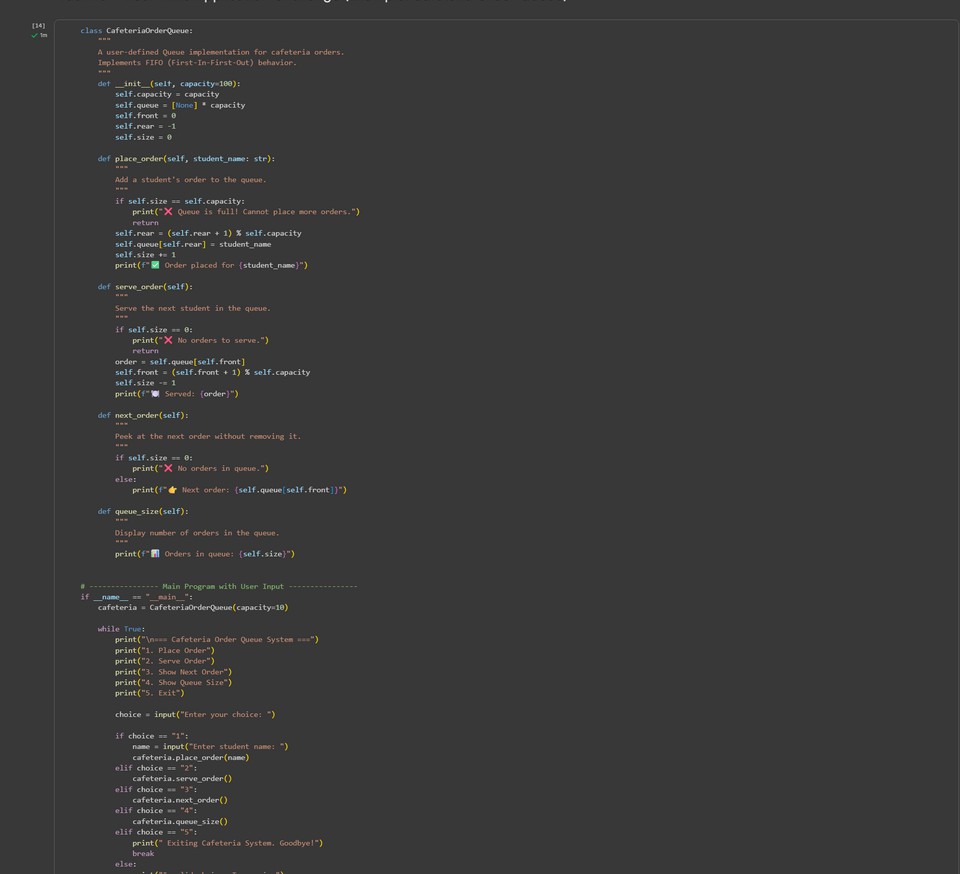
Code :

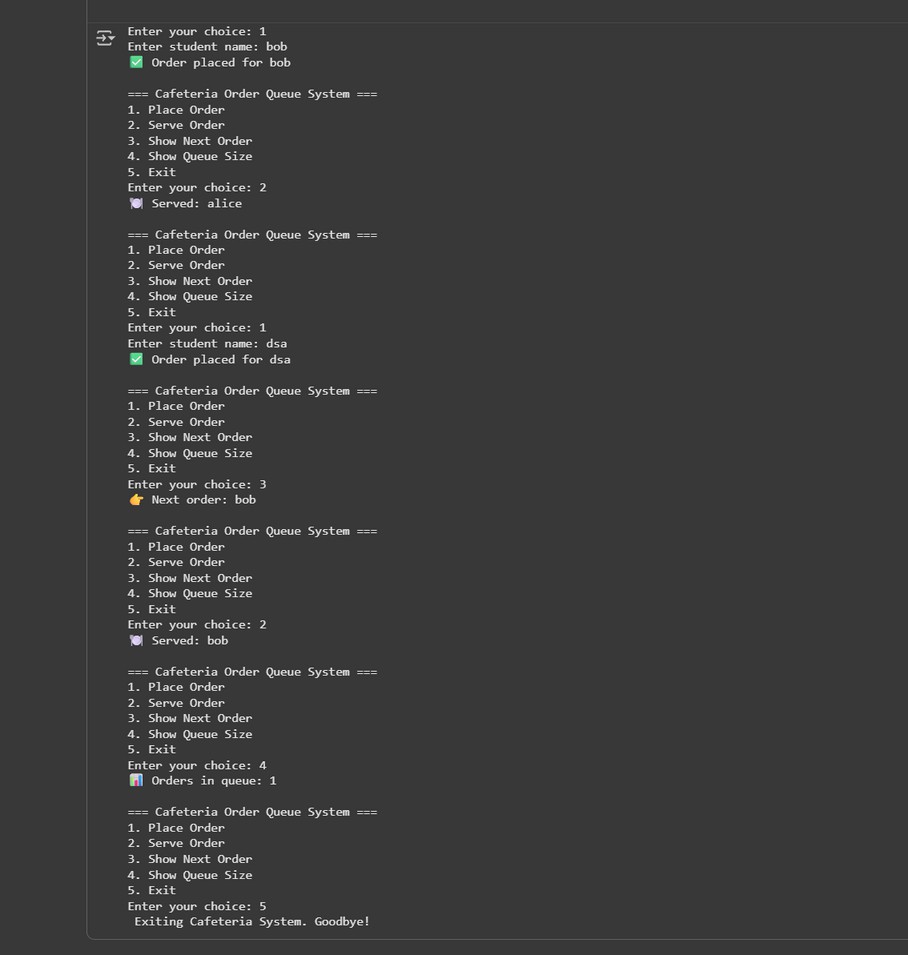




Prompt :

Create a Python menu-driven Queue for cafeteria orders. Include methods: place\_order, serve\_order, next\_order, queue\_size. Use user input to interact.





Observations and Code Explanation

CafeteriaOrderQueue uses circular queue array.

place\_order(student\_name) → adds student; handles full queue. serve\_order() → serves front student; handles empty queue. next\_order() → shows next order without removing. queue\_size() → displays number of orders.

Menu allows user to place, serve, peek, check size, exit interactively.