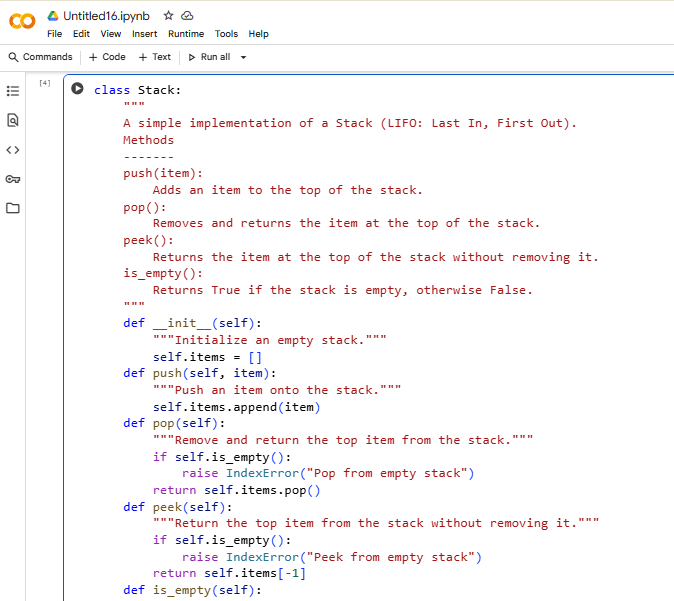
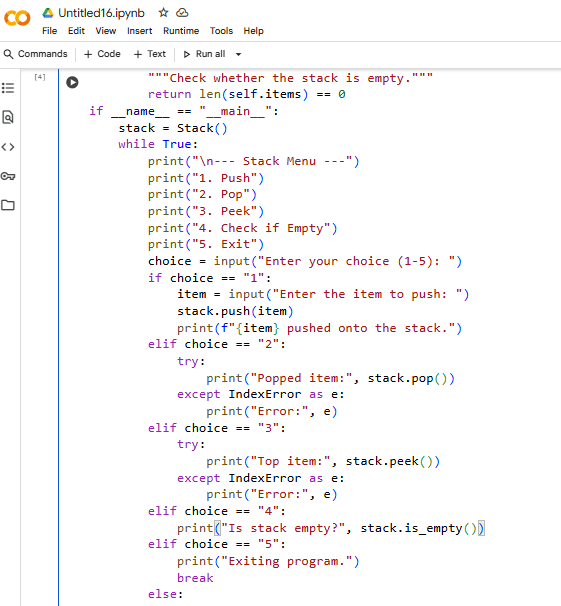


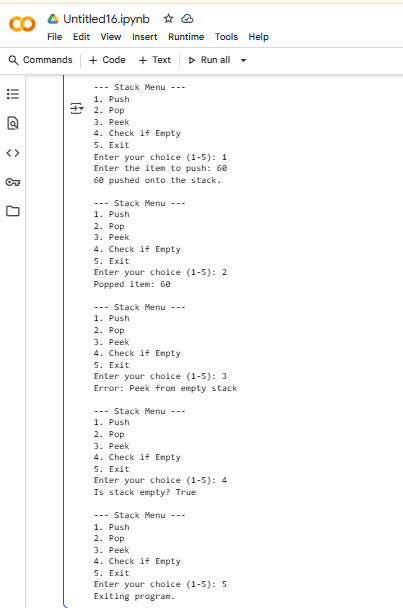
**Prompt :**

Create a Python class named Stack with methods: push, pop, peek, and is\_empty with user input

**Code:**







**observations and code expalnation**

 Defined Stack class with a list to store elements.

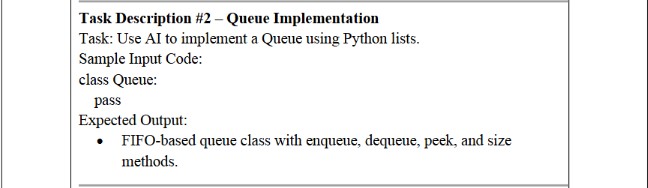
 push(item) → adds an element to the top of stack.

 pop() → removes and returns the top element; handles empty stack.

 peek() → shows top element without removing it.

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

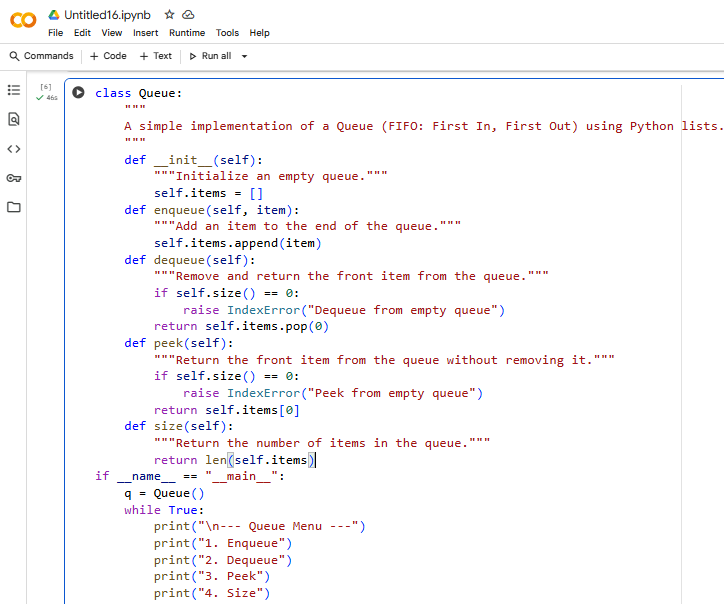
 Interactive code asks for number of elements → pushes elements from user input.

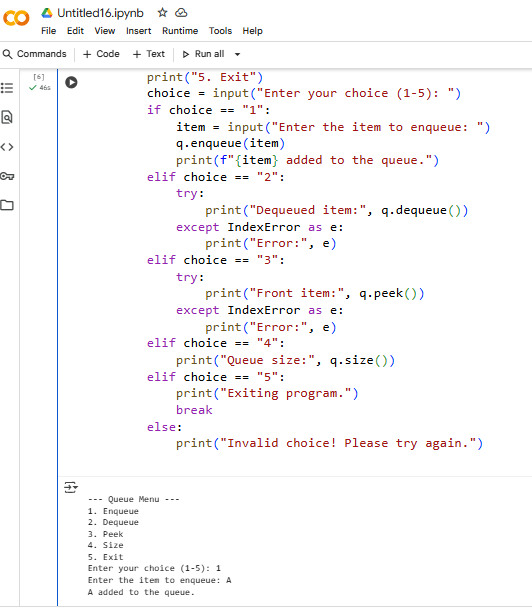


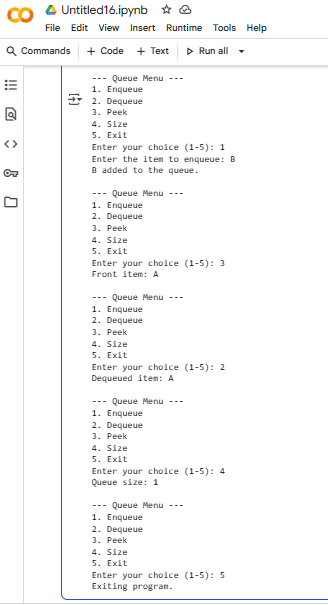
**prompt :**

Create a Python class named Queue using list. Include methods: enqueue, dequeue, peek, and size with user input.

**code:**







**observations and code expalnation :**

 Queue class stores elements in a list.

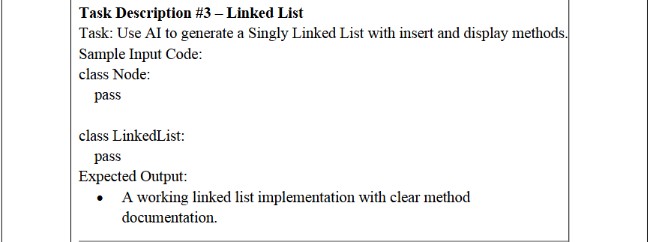
 enqueue(item) → adds element at the rear.

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

 peek() → shows front element.

 size() → returns number of elements.

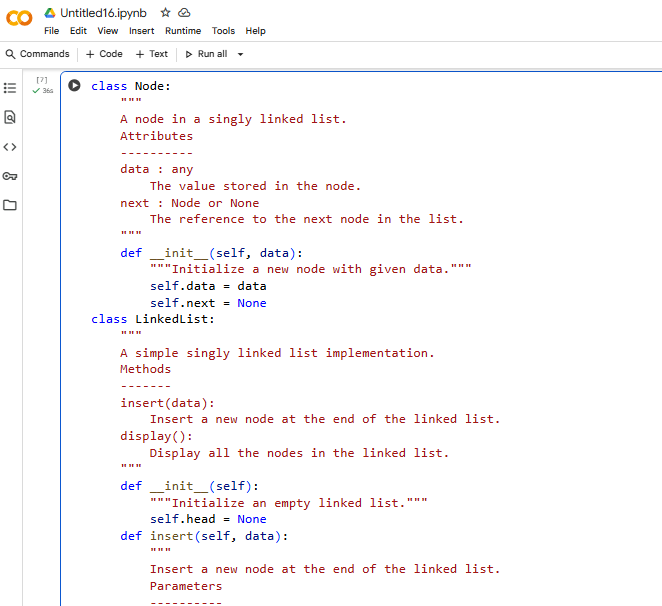
 User input used to enqueue multiple elements; then displays front and dequeued element.

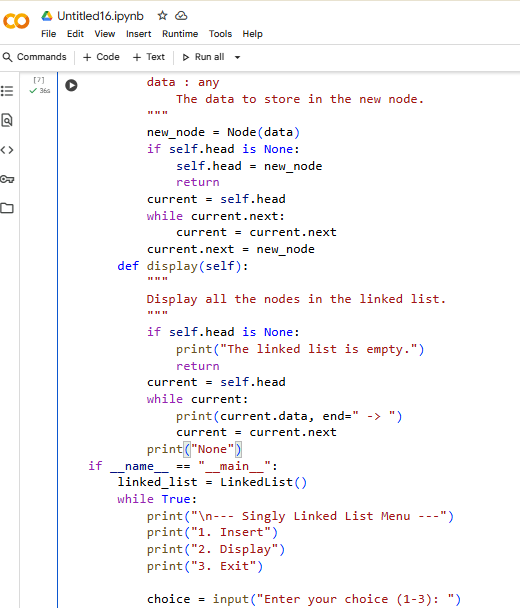


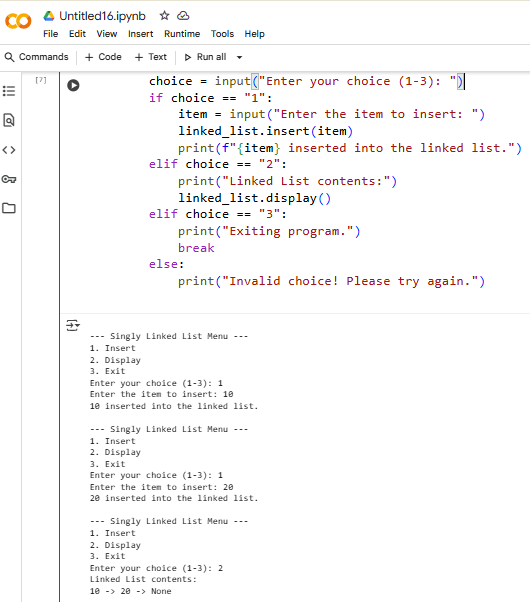
**Prompt :**

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

**Code:**







**Code explanation and observations :**

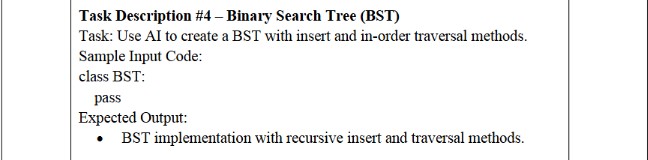
 Node class stores data and next pointer.

 LinkedList class has head pointer.

 insert(data) → adds new node at end.

 display() → prints all nodes in order.

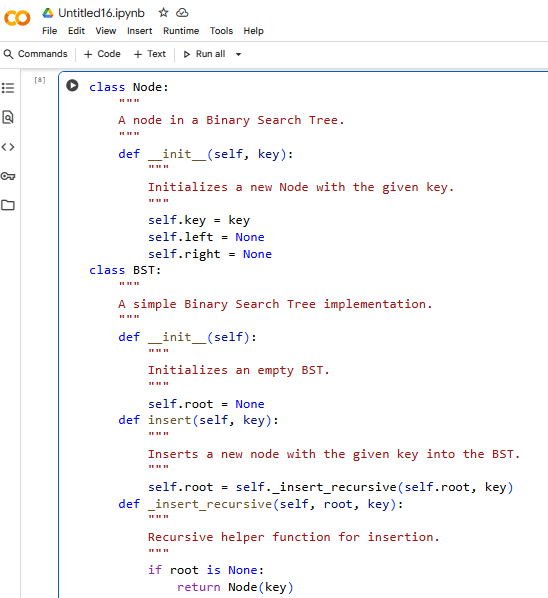
 User inputs number of nodes → program inserts each one and displays list.

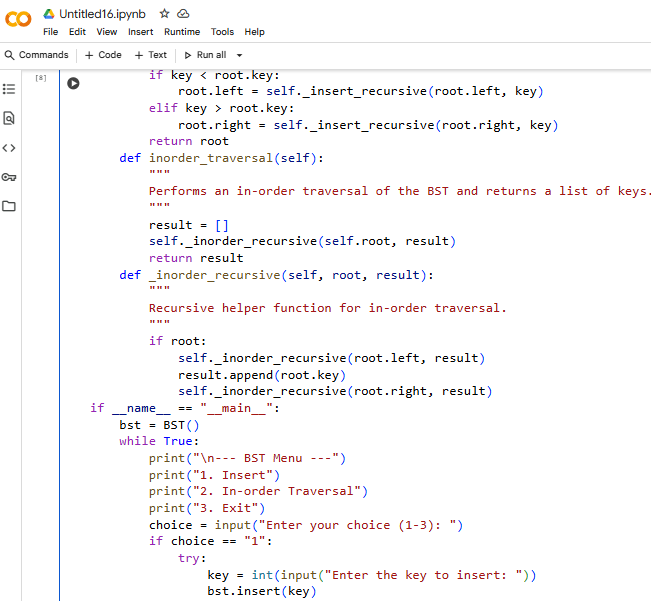


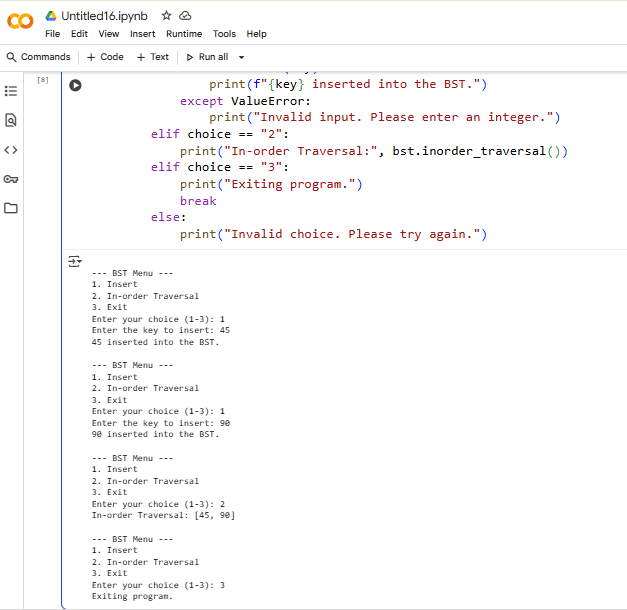
**Prompt :**

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

**Code :**





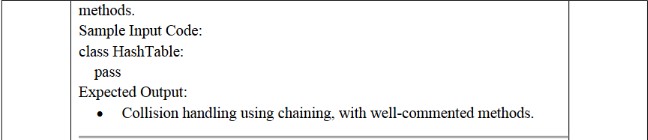


**observations and code expalnation**

BST node has data, left, right.

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

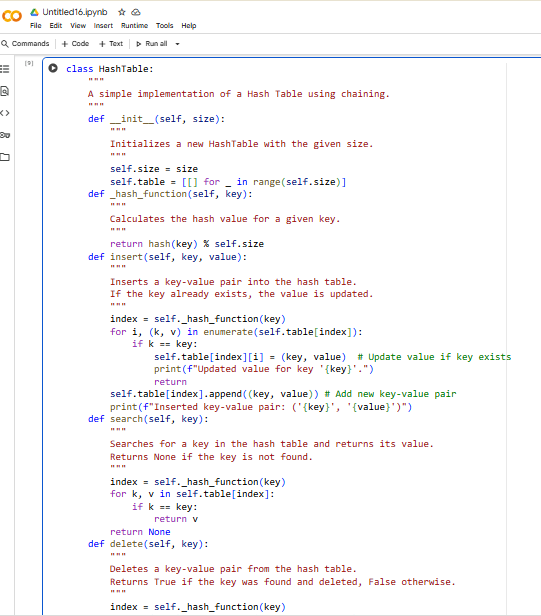
Program asks for root, then number of elements → inserts each. Prints inorder traversal to verify BST structure.

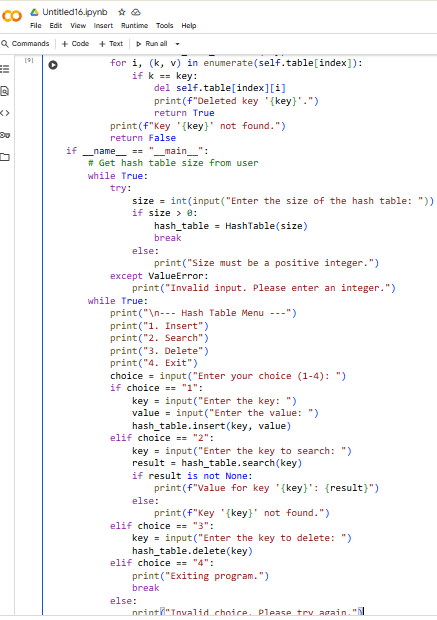


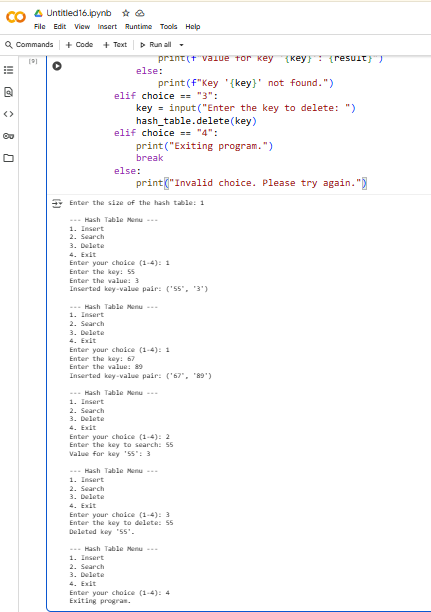
**Prompts** :

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

**Code:**





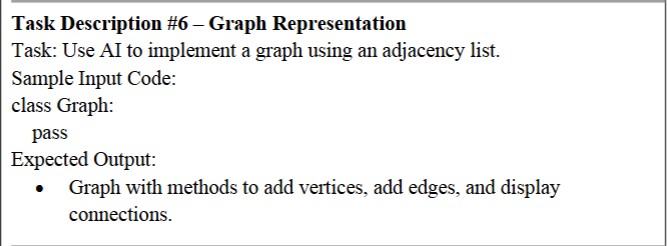


**Observation and code explanation**

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.

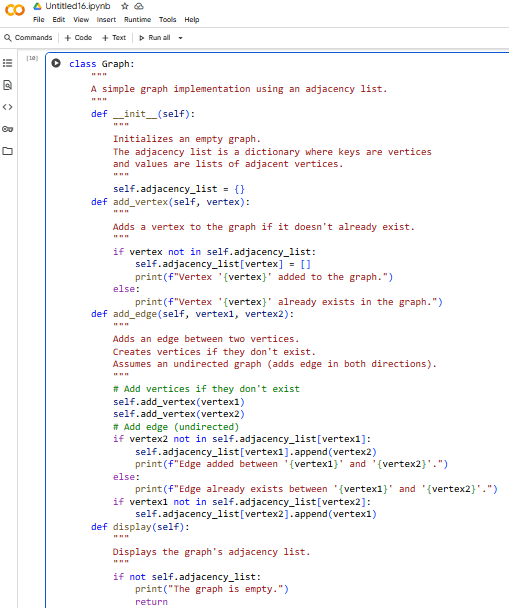
User enters number of key-value pairs → program inserts → searches and prints result.

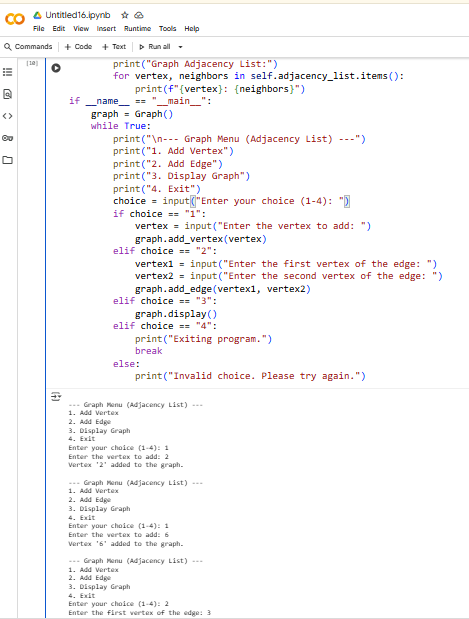


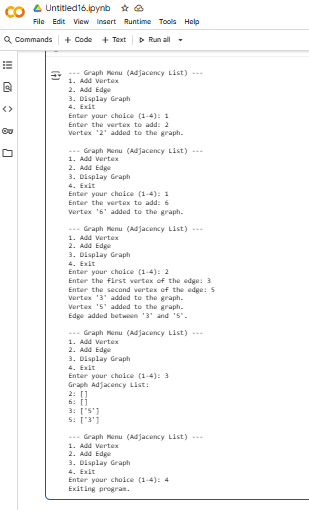
**Prompt :**

Create a Python Graph class using adjacency list. Include methods: add\_vertex, add\_edge, display with user input

**Code:**







**Code explanation and Observation :**

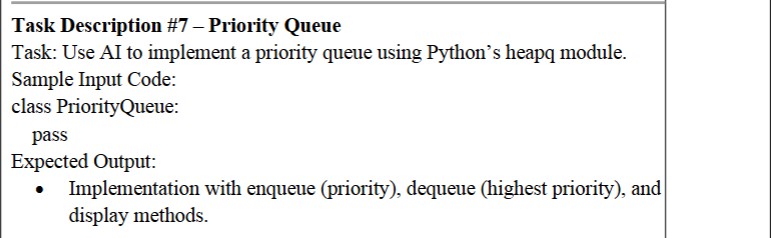
 Dictionary stores adjacency list.

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

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

 display() → prints adjacency list.

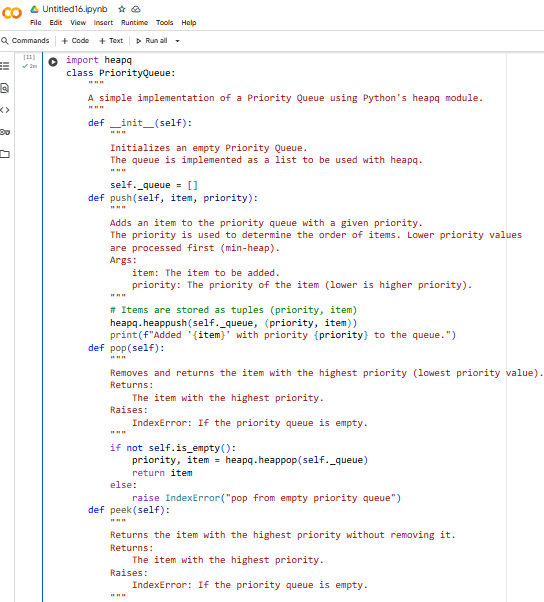
 Program asks number of vertices and edges → user enters each → displays graph.

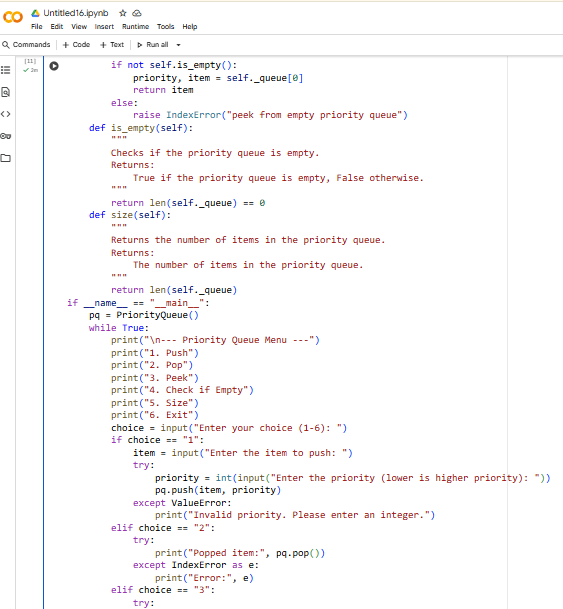


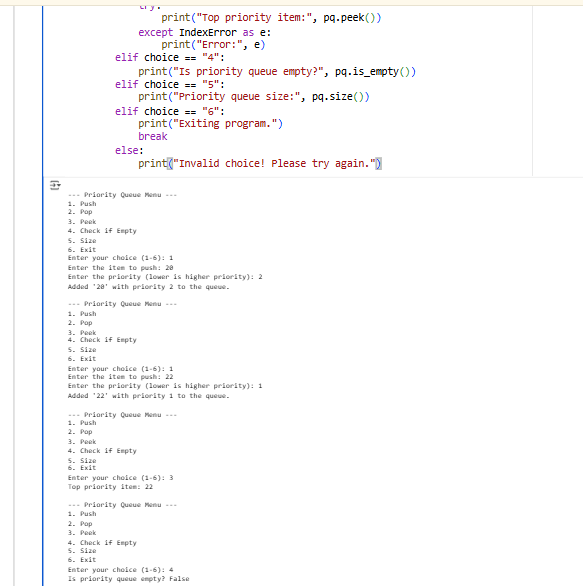
**Prompt :**

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

**Code :**



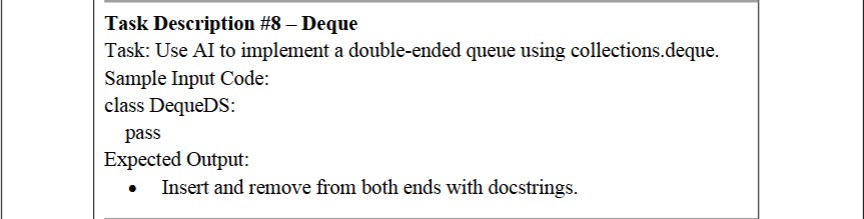




**Observation and code explanation:**

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

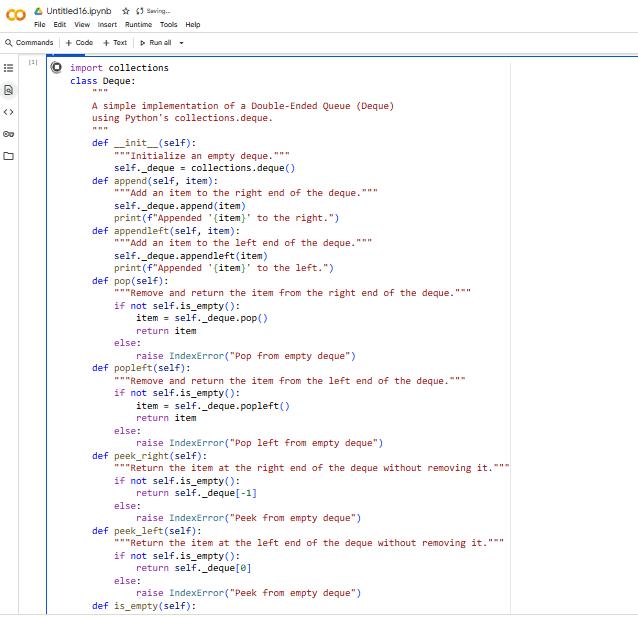
User enters items with priority → program enqueues → dequeues highest priority → displays queue.

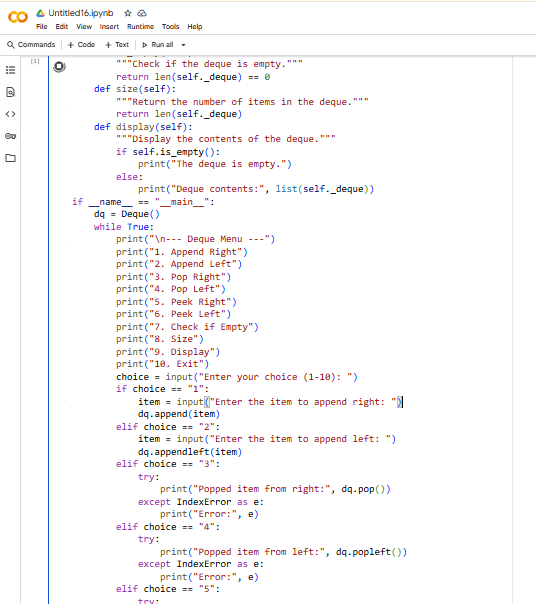


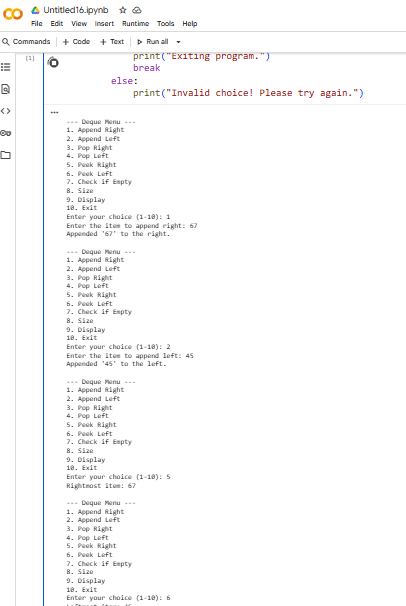
**Prompt :**

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.

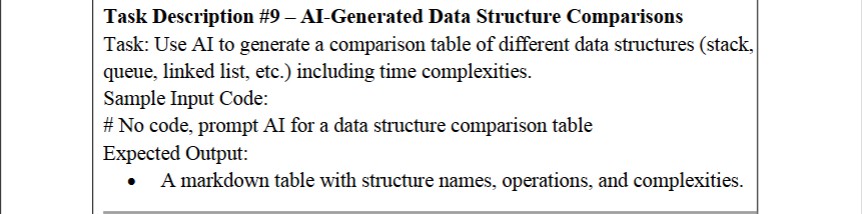
 insert\_front(item) → adds to front.

 insert\_rear(item) → adds to rear.

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

 display() → prints deque.

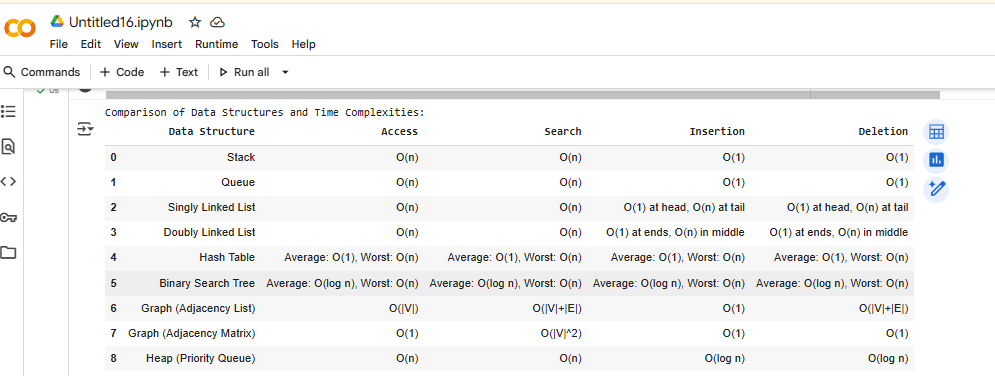
 User inputs number of elements → inserts → removes → displays deque.

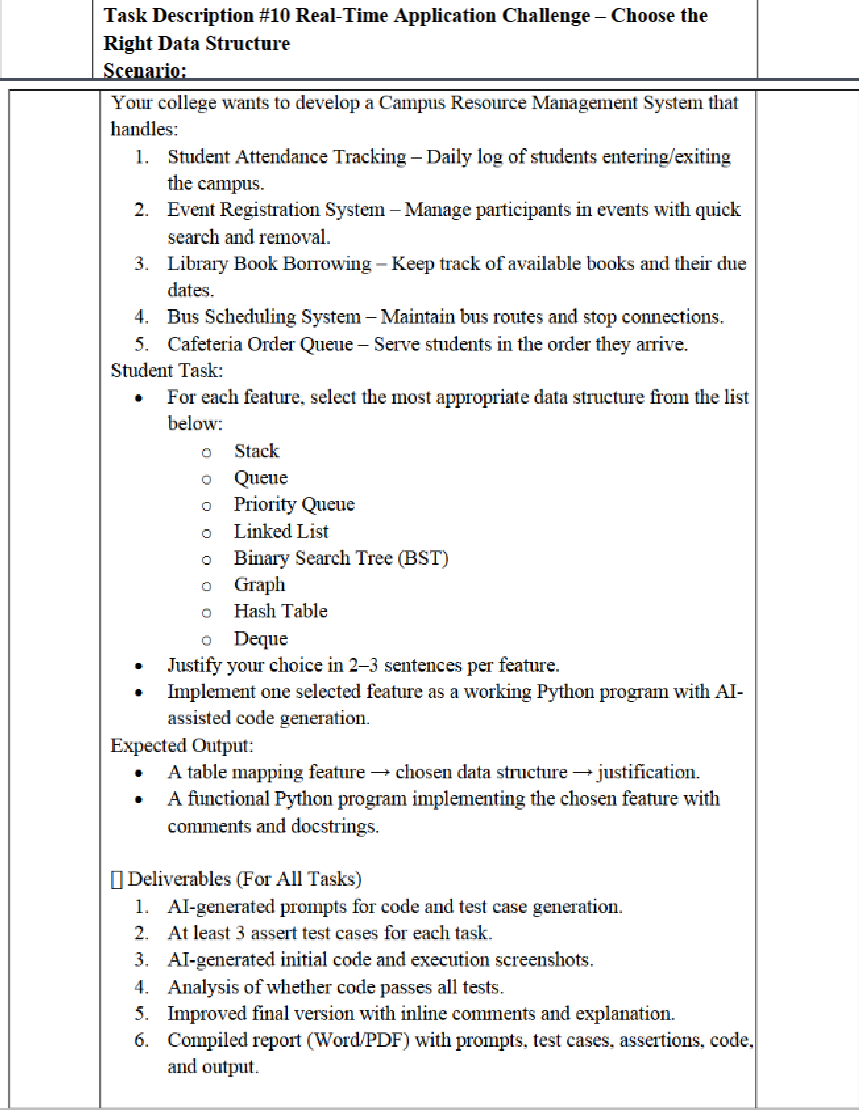


**Prompt :**

Generate a python code for comparison table of different data structures (stack,queue,linked list).

**Table :**

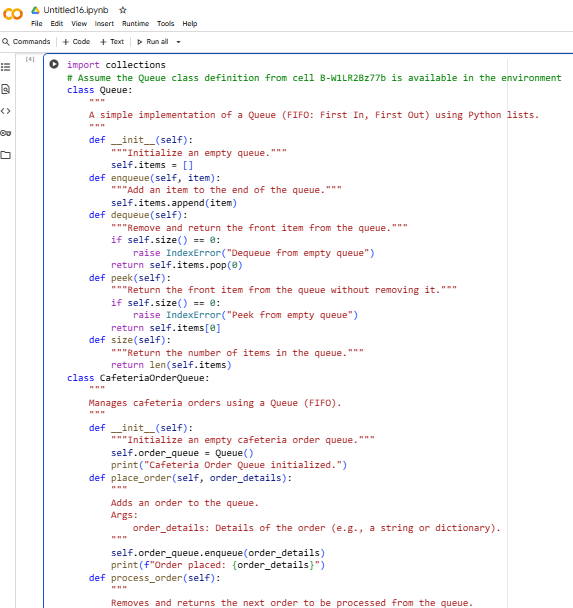


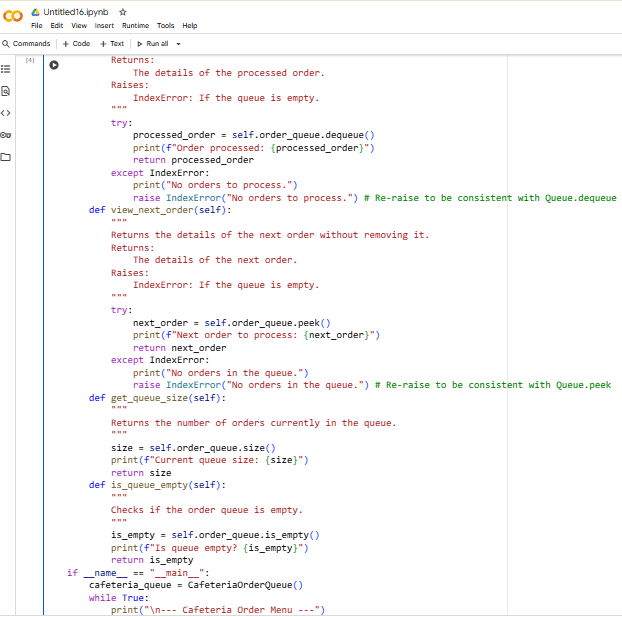


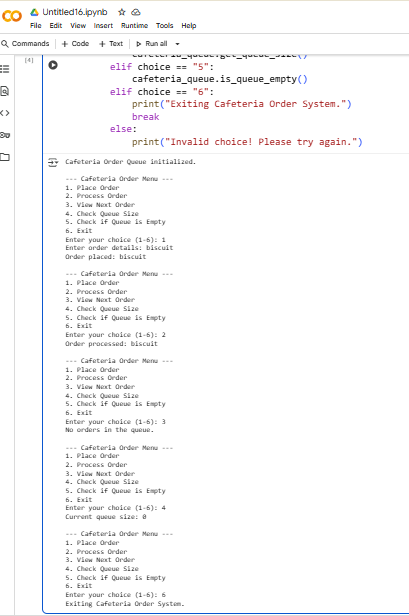
**Prompt :**

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

**Code:**







**Observations and Code Explanation:**

 **Circular Queue:** The system uses a circular array to manage cafeteria orders.

 **Placing Orders:** place\_order(student\_name) adds a student unless the queue is full.

 **Serving Orders:** serve\_order() removes the front order, handling empty queue cases.

 **Peeking & Size:** next\_order() shows the next student without removing; queue\_size() shows pending orders.

 **Interactive Menu:** Users can place, serve, peek, check size, or exit through a menu system.