**Name:- Milind Kailas Tajane**

**Roll No:- CS061**

**Date:-\_\_\_\_\_\_\_\_\_\_\_\_**

**Practical No:1**

# ----------------------------------------------------------------

**AIM:- Write a program to find the sum and product of two matrices using the list data structure.**

**---------------------------------------------------------------------------------------------------------------**

**CODE:-**

def matrix\_sum(matrix1, matrix2):

*# Ensure matrices have the same dimensions*

*if* len(matrix1) *!=* len(matrix2) *or* len(matrix1[0]) *!=* len(matrix2[0]):

*raise* ValueError("Matrices must have the same dimensions to compute the sum.")

*# Add the corresponding elements of the two matrices*

*return* [[matrix1[i][j] *+* matrix2[i][j] *for* j *in* range(len(matrix1[0]))] *for* i *in* range(len(matrix1))]

def matrix\_product(matrix1, matrix2):

*# Ensure the number of columns in matrix1 matches the number of rows in matrix2*

*if* len(matrix1[0]) *!=* len(matrix2):

*raise* ValueError("Number of columns in matrix1 must equal the number of rows in matrix2 to compute the product.")

*# Compute the product of the matrices*

    result *=* [[sum(matrix1[i][k] *\** matrix2[k][j] *for* k *in* range(len(matrix2)))

*for* j *in* range(len(matrix2[0]))] *for* i *in* range(len(matrix1))]

*return* result

*# Example usage*

matrix1 *=* [

    [1, 2, 3],

    [4, 5, 6],

    [7, 8, 9]

]

matrix2 *=* [

    [9, 8, 7],

    [6, 5, 4],

    [3, 2, 1]

]

*# Compute sum and product*

*try*:

    sum\_result *=* matrix\_sum(matrix1, matrix2)

    print("Sum of the matrices:")

*for* row *in* sum\_result:

        print(row)

    product\_result *=* matrix\_product(matrix1, matrix2)

    print("\nProduct of the matrices:")

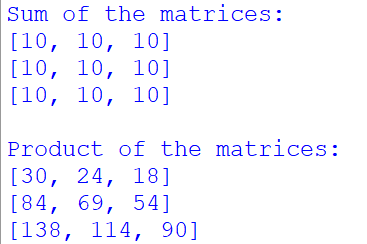
*for* row *in* product\_result:

        print(row)

*except* ValueError *as* e:

    print(e)

**Output:-**



**Name:- Milind Kailas Tajane**

**Roll No:- CS061**

**Date:-\_\_\_\_\_\_\_\_\_\_\_\_**

**Practical No:2**

# ----------------------------------------------------------------

**AIM:- Write a program to implement linked lists (single, doubly, and circular) with functions for adding, deleting, and displaying elements using the linked list data structure.**

**----------------------------------------------------------------------------------------------------------------**

**CODE:-**

*# Singly Linked List Implementation Started*

class Node:

    def \_\_init\_\_(self, data):

        self.data *=* data

        self.next *=* None

class SinglyLinkedList:

    def \_\_init\_\_(self):

        self.head *=* None

    def add(self, data):

        new\_node *=* Node(data)

*if* *not* self.head:

            self.head *=* new\_node

*else*:

            current *=* self.head

*while* current.next:

                current *=* current.next

            current.next *=* new\_node

    def delete(self, data):

*if* *not* self.head:

            print("List is empty.")

*return*

*if* self.head.data *==* data:

            self.head *=* self.head.next

*return*

        current *=* self.head

*while* current.next *and* current.next.data *!=* data:

            current *=* current.next

*if* current.next:

            current.next *=* current.next.next

*else*:

            print("Element not found.")

    def display(self):

        current *=* self.head

*while* current:

            print(current.data, end*=*" -> ")

            current *=* current.next

        print("None")

*# Singly Linked List Implementation End*

*# Doubly Linked List Implementation Started*

class DoublyNode:

    def \_\_init\_\_(self, data):

        self.data *=* data

        self.prev *=* None

        self.next *=* None

class DoublyLinkedList:

    def \_\_init\_\_(self):

        self.head *=* None

    def add(self, data):

        new\_node *=* DoublyNode(data)

*if* *not* self.head:

            self.head *=* new\_node

*else*:

            current *=* self.head

*while* current.next:

                current *=* current.next

            current.next *=* new\_node

            new\_node.prev *=* current

    def delete(self, data):

*if* *not* self.head:

            print("List is empty.")

*return*

*if* self.head.data *==* data:

            self.head *=* self.head.next

*if* self.head:

                self.head.prev *=* None

*return*

        current *=* self.head

*while* current *and* current.data *!=* data:

            current *=* current.next

*if* current:

*if* current.next:

                current.next.prev *=* current.prev

*if* current.prev:

                current.prev.next *=* current.next

*else*:

            print("Element not found.")

    def display(self):

        current *=* self.head

*while* current:

            print(current.data, end*=*" <-> ")

            current *=* current.next

        print("None")

*# Doubly linked List Implementation End*

*# Circular Linked List Implementation Started*

class CircularNode:

    def \_\_init\_\_(self, data):

        self.data *=* data

        self.next *=* None

class CircularLinkedList:

    def \_\_init\_\_(self):

        self.head *=* None

    def add(self, data):

        new\_node *=* CircularNode(data)

*if* *not* self.head:

            self.head *=* new\_node

            self.head.next *=* self.head

*else*:

            current *=* self.head

*while* current.next *!=* self.head:

                current *=* current.next

            current.next *=* new\_node

            new\_node.next *=* self.head

    def delete(self, data):

*if* *not* self.head:

            print("List is empty.")

*return*

*if* self.head.data *==* data:

*if* self.head.next *==* self.head:  *# Only one element*

                self.head *=* None

*else*:

                current *=* self.head

*while* current.next *!=* self.head:

                    current *=* current.next

                current.next *=* self.head.next

                self.head *=* self.head.next

*return*

        current *=* self.head

*while* current.next *!=* self.head *and* current.next.data *!=* data:

            current *=* current.next

*if* current.next.data *==* data:

            current.next *=* current.next.next

*else*:

            print("Element not found.")

    def display(self):

*if* *not* self.head:

            print("List is empty.")

*return*

        current *=* self.head

*while* True:

            print(current.data, end*=*" -> ")

            current *=* current.next

*if* current *==* self.head:

*break*

        print("(head)")

*# Circular Linked List Implementation End*

*# Singly Linked List*

print("Singly Linked List:")

sll *=* SinglyLinkedList()

sll.add(1)

sll.add(2)

sll.add(3)

sll.display()

sll.delete(2)

sll.display()

*# Doubly Linked List*

print("\nDoubly Linked List:")

dll *=* DoublyLinkedList()

dll.add(1)

dll.add(2)

dll.add(3)

dll.display()

dll.delete(2)

dll.display()

*# Circular Linked List*

print("\nCircular Linked List:")

cll *=* CircularLinkedList()

cll.add(1)

cll.add(2)

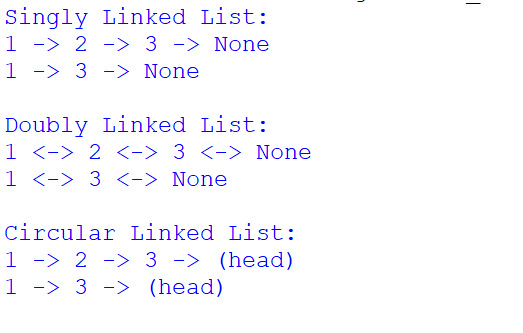
cll.add(3)

cll.display()

cll.delete(2)

cll.display()

**Output:-**

****

**Name:- Milind Kailas Tajane**

**Roll No:- CS061**

**Date:-\_\_\_\_\_\_\_\_\_\_\_\_**

**Practical No: 3**

# ----------------------------------------------------------------

**AIM:- Write a program to implement binary search on a sorted list using the array data structure.**

**----------------------------------------------------------------------------------------------------------------**

**CODE:-**

def binary\_search(arr, target):

    """

    Perform binary search on a sorted array.

    :param arr: Sorted list of elements

    :param target: Element to search for

    :return: Index of the target element or -1 if not found

    """

    left, right *=* 0, len(arr) *-* 1

*while* left *<=* right:

        mid *=* (left *+* right) *//* 2  *# Find the middle index*

*if* arr[mid] *==* target:

*return* mid  *# Target found*

*elif* arr[mid] *<* target:

            left *=* mid *+* 1  *# Target is in the right half*

*else*:

            right *=* mid *-* 1  *# Target is in the left half*

*return* *-*1  *# Target not found*

*# Example usage*

sorted\_list *=* [1, 3, 5, 7, 9, 11, 13, 15]

target *=* 7

result *=* binary\_search(sorted\_list, target)

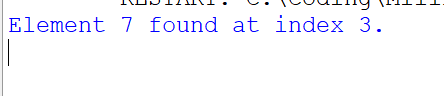
*if* result *!=* *-*1:

    print(f"Element {target} found at index {result}.")

*else*:

    print(f"Element {target} not found in the list.")

**Output:-**



**Name:- Milind Kailas Tajane**

**Roll No:- CS061**

**Date:-\_\_\_\_\_\_\_\_\_\_\_\_**

**Practical No: 4**

# ----------------------------------------------------------------

**AIM:- Write a program to implement insertion sort, selection sort, and bubble sort algorithms using the array/list data structure.**

**----------------------------------------------------------------------------------------------------------------**

**CODE:-**

*# Insertion Sort*

def insertion\_sort(arr):

*for* i *in* range(1, len(arr)):

        key *=* arr[i]

        j *=* i *-* 1

*# Move elements greater than key to one position ahead*

*while* j *>=* 0 *and* arr[j] *>* key:

            arr[j *+* 1] *=* arr[j]

            j *-=* 1

        arr[j *+* 1] *=* key

*return* arr

*# Selection Sort*

def selection\_sort(arr):

*for* i *in* range(len(arr)):

        min\_index *=* i

*# Find the minimum element in the remaining unsorted portion*

*for* j *in* range(i *+* 1, len(arr)):

*if* arr[j] *<* arr[min\_index]:

                min\_index *=* j

*# Swap the found minimum element with the first element of the unsorted portion*

        arr[i], arr[min\_index] *=* arr[min\_index], arr[i]

*return* arr

*# Bubble Sort*

def bubble\_sort(arr):

    n *=* len(arr)

*for* i *in* range(n):

*# Last i elements are already sorted*

*for* j *in* range(0, n *-* i *-* 1):

*if* arr[j] *>* arr[j *+* 1]:

*# Swap if the current element is greater than the next*

                arr[j], arr[j *+* 1] *=* arr[j *+* 1], arr[j]

*return* arr

*# Example usage*

unsorted\_list *=* [64, 25, 12, 22, 11]

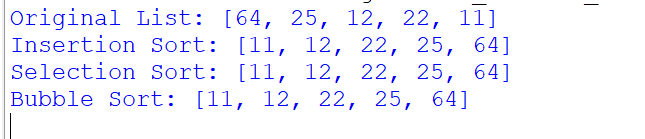
print("Original List:", unsorted\_list)

print("Insertion Sort:", insertion\_sort(unsorted\_list.copy()))

print("Selection Sort:", selection\_sort(unsorted\_list.copy()))

print("Bubble Sort:", bubble\_sort(unsorted\_list.copy()))

**Output:-**



**Name:- Milind Kailas Tajane**

**Roll No:- CS061**

**Date:-\_\_\_\_\_\_\_\_\_\_\_\_**

**Practical No: 5**

# ----------------------------------------------------------------

**AIM:- Write a program to implement a binary search tree (BST) with operations for insertion, deletion, and in-order traversal using the tree data structure.**

**----------------------------------------------------------------------------------------------------------------**

**CODE:-**

class Node:

    """Node class for Binary Search Tree."""

    def \_\_init\_\_(self, key):

        self.key *=* key

        self.left *=* None

        self.right *=* None

class BinarySearchTree:

    """Binary Search Tree implementation."""

    def \_\_init\_\_(self):

        self.root *=* None

    def insert(self, key):

        """Insert a new key into the BST."""

*if* self.root *is* None:

            self.root *=* Node(key)

*else*:

            self.\_insert(self.root, key)

    def \_insert(self, current, key):

*if* key *<* current.key:

*if* current.left *is* None:

                current.left *=* Node(key)

*else*:

                self.\_insert(current.left, key)

*else*:

*if* current.right *is* None:

                current.right *=* Node(key)

*else*:

                self.\_insert(current.right, key)

    def delete(self, key):

        """Delete a key from the BST."""

        self.root *=* self.\_delete(self.root, key)

    def \_delete(self, current, key):

*if* current *is* None:

*return* current

*if* key *<* current.key:

            current.left *=* self.\_delete(current.left, key)

*elif* key *>* current.key:

            current.right *=* self.\_delete(current.right, key)

*else*:

*# Node with only one child or no child*

*if* current.left *is* None:

*return* current.right

*elif* current.right *is* None:

*return* current.left

*# Node with two children: Get the inorder successor (smallest in the right subtree)*

            min\_larger\_node *=* self.\_find\_min(current.right)

            current.key *=* min\_larger\_node.key

            current.right *=* self.\_delete(current.right, min\_larger\_node.key)

*return* current

    def \_find\_min(self, node):

        """Find the node with the smallest key."""

*while* node.left *is* *not* None:

            node *=* node.left

*return* node

    def in\_order\_traversal(self):

        """Perform in-order traversal of the BST."""

*return* self.\_in\_order\_traversal(self.root, [])

    def \_in\_order\_traversal(self, current, result):

*if* current *is* *not* None:

            self.\_in\_order\_traversal(current.left, result)

            result.append(current.key)

            self.\_in\_order\_traversal(current.right, result)

*return* result

*# Example usage*

bst *=* BinarySearchTree()

*# Insert elements*

bst.insert(50)

bst.insert(30)

bst.insert(20)

bst.insert(40)

bst.insert(70)

bst.insert(60)

bst.insert(80)

print("In-order traversal after insertion:", bst.in\_order\_traversal())

*# Delete elements*

bst.delete(20)

print("In-order traversal after deleting 20:", bst.in\_order\_traversal())

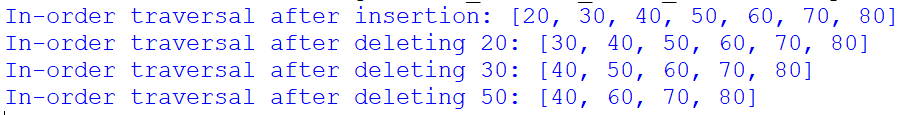
bst.delete(30)

print("In-order traversal after deleting 30:", bst.in\_order\_traversal())

bst.delete(50)

print("In-order traversal after deleting 50:", bst.in\_order\_traversal())

**Output:-**



**Name:- Milind Kailas Tajane**

**Roll No:- CS061**

**Date:-\_\_\_\_\_\_\_\_\_\_\_\_**

**Practical No: 6**

# ----------------------------------------------------------------

**AIM:- Write a program to implement a graph using an adjacency list and perform both depth-first search (DFS) and breadth-first search (BFS) using the graph data structure.**

**----------------------------------------------------------------------------------------------------------------**

**CODE:-**

*from* collections *import* deque, defaultdict

class Graph:

    """Graph implemented using an adjacency list."""

    def \_\_init\_\_(self):

        self.adj\_list *=* defaultdict(list)

    def add\_edge(self, u, v):

        """Add an edge to the graph (u -> v)."""

        self.adj\_list[u].append(v)

    def dfs(self, start):

        """Perform Depth-First Search (DFS)."""

        visited *=* set()

        result *=* []

        def dfs\_recursive(node):

*if* node *not* *in* visited:

                visited.add(node)

                result.append(node)

*for* neighbor *in* self.adj\_list[node]:

                    dfs\_recursive(neighbor)

        dfs\_recursive(start)

*return* result

    def bfs(self, start):

        """Perform Breadth-First Search (BFS)."""

        visited *=* set()

        queue *=* deque([start])

        result *=* []

*while* queue:

            node *=* queue.popleft()

*if* node *not* *in* visited:

                visited.add(node)

                result.append(node)

*for* neighbor *in* self.adj\_list[node]:

*if* neighbor *not* *in* visited:

                        queue.append(neighbor)

*return* result

    def display\_graph(self):

        """Display the adjacency list of the graph."""

*for* node, neighbors *in* self.adj\_list.items():

            print(f"{node} -> {', '.join(map(str, neighbors))}")

*# Example usage*

graph *=* Graph()

graph.add\_edge(1, 2)

graph.add\_edge(1, 3)

graph.add\_edge(2, 4)

graph.add\_edge(2, 5)

graph.add\_edge(3, 6)

graph.add\_edge(3, 7)

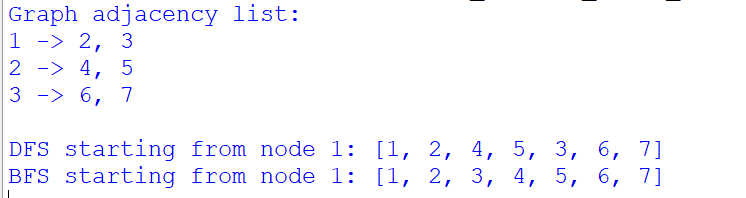
print("Graph adjacency list:")

graph.display\_graph()

print("\nDFS starting from node 1:", graph.dfs(1))

print("BFS starting from node 1:", graph.bfs(1))

**Output:-**



**Name:- Milind Kailas Tajane**

**Roll No:- CS061**

**Date:-\_\_\_\_\_\_\_\_\_\_\_\_**

**Practical No:7**

# ----------------------------------------------------------------

**AIM:- Write a program to implement a priority queue using the heap data structure (min-heap or max-heap).**

**---------------------------------------------------------------------------------------------------------------**

**CODE:-**

*import heapq*

*class PriorityQueue:*

*"""Priority Queue implemented using a heap (min-heap)."""*

*def \_\_init\_\_(self):*

*self.heap = []*

*def push(self, item, priority):*

*"""Insert an item with a given priority into the priority queue."""*

*heapq.heappush(self.heap, (priority, item)) # Push as a tuple (priority, item)*

*def pop(self):*

*"""Remove and return the item with the highest priority (lowest value)."""*

*if not self.is\_empty():*

*return heapq.heappop(self.heap)[1] # Return only the item*

*raise IndexError("Pop from an empty priority queue")*

*def peek(self):*

*"""Return the item with the highest priority without removing it."""*

*if not self.is\_empty():*

*return self.heap[0][1] # Return only the item*

*raise IndexError("Peek from an empty priority queue")*

*def is\_empty(self):*

*"""Check if the priority queue is empty."""*

*return len(self.heap) == 0*

*def display(self):*

*"""Display the contents of the priority queue."""*

*print("Priority Queue contents:", self.heap)*

*# Example usage*

*pq = PriorityQueue()*

*# Insert items with priorities*

*pq.push("Task A", 3)*

*pq.push("Task B", 1)*

*pq.push("Task C", 2)*

*print("After adding tasks:")*

*pq.display()*

*# Get the item with the highest priority (lowest value)*

*print("\nPeek:", pq.peek())*

*# Remove items based on priority*

*print("\nPop:", pq.pop())*

*print("Pop:", pq.pop())*

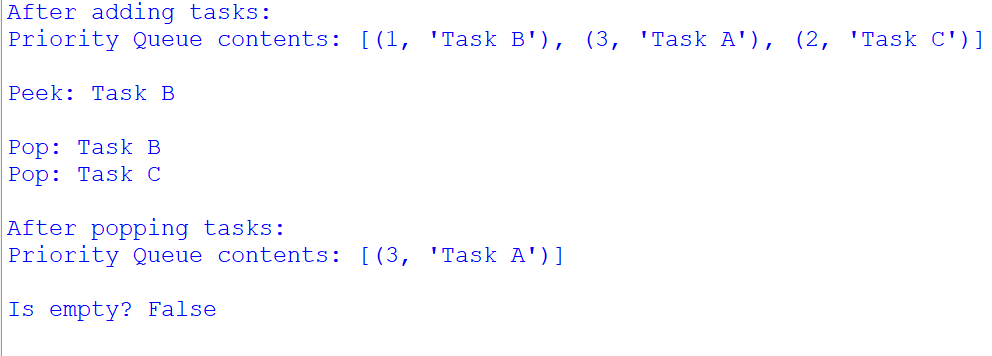
*print("\nAfter popping tasks:")*

*pq.display()*

*# Check if the queue is empty*

*print("\nIs empty?", pq.is\_empty())*

**Output:-**

****

**Name:- Milind Kailas Tajane**

**Roll No:- CS061**

**Date:-\_\_\_\_\_\_\_\_\_\_\_\_**

**Practical No:8**

# ----------------------------------------------------------------

**AIM:- Write a program to implement a hash table with collision handling using chaining, demonstrating the hash table data structure.**

**----------------------------------------------------------------------------------------------------------------**

**CODE:-**

class HashTable:

    """Hash Table implementation with chaining for collision handling."""

    def \_\_init\_\_(self, size):

        self.size *=* size

        self.table *=* [[] *for* \_ *in* range(size)]

    def \_hash(self, key):

        """Compute the hash value of a key."""

*return* hash(key) *%* self.size

    def insert(self, key, value):

        """Insert a key-value pair into the hash table."""

        index *=* self.\_hash(key)

*# Check if the key already exists in the chain*

*for* pair *in* self.table[index]:

*if* pair[0] *==* key:

                pair[1] *=* value  *# Update existing key*

*return*

*# If not, append a new key-value pair*

        self.table[index].append([key, value])

    def search(self, key):

        """Search for a value by its key."""

        index *=* self.\_hash(key)

*for* pair *in* self.table[index]:

*if* pair[0] *==* key:

*return* pair[1]

*return* None  *# Key not found*

    def delete(self, key):

        """Remove a key-value pair from the hash table."""

        index *=* self.\_hash(key)

*for* i, pair *in* enumerate(self.table[index]):

*if* pair[0] *==* key:

*del* self.table[index][i]

*return* True

*return* False  *# Key not found*

    def display(self):

        """Display the contents of the hash table."""

*for* i, chain *in* enumerate(self.table):

            print(f"Index {i}: {chain}")

*# Example usage*

hash\_table *=* HashTable(5)

*# Insert key-value pairs*

hash\_table.insert("Alice", 25)

hash\_table.insert("Bob", 30)

hash\_table.insert("Charlie", 35)

hash\_table.insert("David", 40)

hash\_table.insert("Eve", 45)  *# May collide with another key*

print("Hash Table after insertion:")

hash\_table.display()

*# Search for keys*

print("\nSearch results:")

print("Alice:", hash\_table.search("Alice"))

print("Bob:", hash\_table.search("Bob"))

print("Zara:", hash\_table.search("Zara"))  *# Key not present*

*# Delete a key*

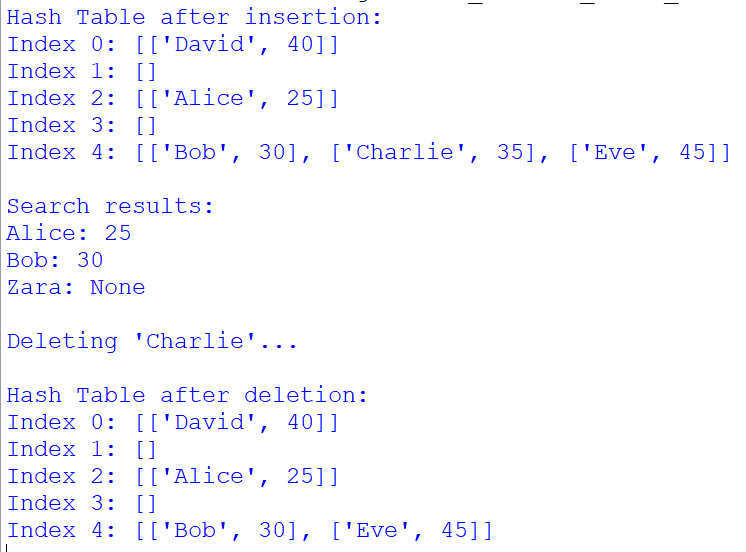
print("\nDeleting 'Charlie'...")

hash\_table.delete("Charlie")

print("\nHash Table after deletion:")

hash\_table.display()

**Output:-**



**Name:- Milind Kailas Tajane**

**Roll No:- CS061**

**Date:-\_\_\_\_\_\_\_\_\_\_\_\_**

**Practical No: 9**

# ----------------------------------------------------------------

**AIM:- Write a program to implement a circular queue using the queue data structure implemented with a list.**

**----------------------------------------------------------------------------------------------------------------**

**CODE:-**

class CircularQueue:

    """Circular Queue implementation using a list."""

    def \_\_init\_\_(self, size):

        self.size *=* size

        self.queue *=* [None] *\** size

        self.front *=* *-*1

        self.rear *=* *-*1

    def enqueue(self, value):

        """Add an element to the queue."""

*if* (self.rear *+* 1) *%* self.size *==* self.front:

            print("Queue is full!")

*elif* self.front *==* *-*1:  *# First element being added*

            self.front *=* self.rear *=* 0

            self.queue[self.rear] *=* value

*else*:

            self.rear *=* (self.rear *+* 1) *%* self.size

            self.queue[self.rear] *=* value

    def dequeue(self):

        """Remove an element from the queue."""

*if* self.front *==* *-*1:

            print("Queue is empty!")

*return* None

*elif* self.front *==* self.rear:  *# Only one element left*

            value *=* self.queue[self.front]

            self.front *=* self.rear *=* *-*1

*else*:

            value *=* self.queue[self.front]

            self.front *=* (self.front *+* 1) *%* self.size

*return* value

    def display(self):

        """Display the elements of the queue."""

*if* self.front *==* *-*1:

            print("Queue is empty!")

*return*

        print("Queue elements:")

        i *=* self.front

*while* True:

            print(self.queue[i], end*=*" ")

*if* i *==* self.rear:

*break*

            i *=* (i *+* 1) *%* self.size

        print()

    def is\_empty(self):

        """Check if the queue is empty."""

*return* self.front *==* *-*1

    def is\_full(self):

        """Check if the queue is full."""

*return* (self.rear *+* 1) *%* self.size *==* self.front

*# Example usage*

cq *=* CircularQueue(5)

*# Enqueue elements*

cq.enqueue(10)

cq.enqueue(20)

cq.enqueue(30)

cq.enqueue(40)

print("\nQueue after enqueues:")

cq.display()

*# Dequeue elements*

print("\nDequeue:", cq.dequeue())

print("Dequeue:", cq.dequeue())

print("\nQueue after dequeues:")

cq.display()

*# Add more elements*

cq.enqueue(50)

cq.enqueue(60)

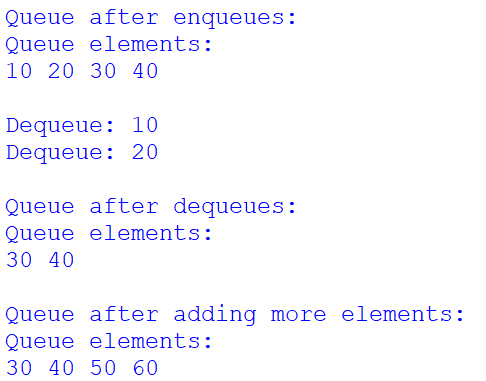
print("\nQueue after adding more elements:")

cq.display()

*# Try to add to a full queue*

cq.enqueue(70)

**Output:-**

****

**Name:- Milind Kailas Tajane**

**Roll No:- CS061**

**Date:-\_\_\_\_\_\_\_\_\_\_\_\_**

**Practical No: 10**

# ----------------------------------------------------------------

**AIM:- Write a program to find the largest and smallest elements in an array using the array/list data structure.**

**----------------------------------------------------------------------------------------------------------------**

**CODE:-**

def find\_largest\_and\_smallest(arr):

    """Find the largest and smallest elements in the array."""

*if* *not* arr:

*return* None, None  *# If the array is empty, return None*

    largest *=* smallest *=* arr[0]  *# Initialize both largest and smallest to the first element*

*for* num *in* arr:

*if* num *>* largest:

            largest *=* num

*if* num *<* smallest:

            smallest *=* num

*return* largest, smallest

*# Example usage*

arr *=* [5, 3, 8, 1, 9, 2, 6]

largest, smallest *=* find\_largest\_and\_smallest(arr)

print(f"Largest element: {largest}")

print(f"Smallest element: {smallest}")

**Output:-**

