DAY-5:

PROGRAM:

Singly linked list:

class Node:

def \_\_init\_\_(self, data):

self.data = data

self.next = None

class LinkedList:

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

self.head = None # starting point of LinkedList

def insert(self, data):

new\_node = Node(data)

if self.head is None:

self.head = new\_node

else:

temp = self.head

while temp.next:

temp = temp.next # move temp for traversal

temp.next = new\_node

def display(self):

temp = self.head

while temp:

print(temp.data, end=" -> ")

temp = temp.next

print("None")

def sum\_of\_nodes(self): # Better name for the function

temp = self.head

total = 0

while temp:

total += temp.data

temp = temp.next

return total

# Test the LinkedList

if \_\_name\_\_ == "\_\_main\_\_":

ll = LinkedList()

ll.insert(10)

ll.insert(20)

ll.insert(30)

ll.display()

print("Sum of nodes:", ll.sum\_of\_nodes())

OUTPUT:  
10 -> 20 -> 30 -> None

Sum of nodes: 60

DELETION:

PROGRAM:  
class Node:

def \_\_init\_\_(self, data):

self.data = data

self.next = None

class LinkedList:

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

self.head = None

def insert\_beginning(self, data):

new\_node = Node(data)

new\_node.next = self.head

self.head = new\_node

def delete\_beginning(self):

if self.head is None:

print("List is empty")

return

self.head = self.head.next

def insert\_position(self, pos, data):

new\_node = Node(data)

if pos == 0:

self.insert\_beginning(data)

return

temp = self.head

for \_ in range(pos - 1):

if temp is None:

print("Position out of bounds")

return

temp = temp.next

if temp is None:

print("Position out of bounds")

return

new\_node.next = temp.next

temp.next = new\_node

def delete\_value(self, value):

if self.head is None:

print("List is empty")

return

if self.head.data == value:

self.head = self.head.next

return

temp = self.head

while temp.next and temp.next.data != value:

temp = temp.next

if temp.next is None:

print("Value not found")

return

temp.next = temp.next.next

def display(self):

temp = self.head

while temp:

print(temp.data, end=" -> ")

temp = temp.next

print("None")

def count(self):

temp = self.head

c = 0

while temp:

c += 1

temp = temp.next

return c

ll = LinkedList()

ll.insert\_beginning(10)

ll.insert\_beginning(20)

ll.insert\_position(1, 15)

ll.display()

ll.delete\_value(15)

ll.display()

print("Total nodes:", ll.count())

OUTPUT:

20 -> 15 -> 10 -> None

20 -> 10 -> None

Total nodes: 2

DOUBLY LINKED LIST:

PROGRAM:

class Node:

def \_\_init\_\_(self, data):

self.prev = None

self.next = None

self.data = data

class DoublyLinkedList:

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

self.head = None

def insert\_beginning(self, data):

new\_node = Node(data)

if self.head is None:

self.head = new\_node

else:

new\_node.next = self.head

self.head.prev = new\_node

self.head = new\_node

def delete\_beginning(self):

if self.head is None:

print("List is empty")

return

if self.head.next is None:

self.head = None

else:

self.head = self.head.next

self.head.prev = None

def delete\_end(self):

if self.head is None:

print("List is empty")

return

if self.head.next is None:

self.head = None

else:

temp = self.head

while temp.next:

temp = temp.next

temp.prev.next = None

def display(self):

temp = self.head

while temp:

print(temp.data, end=" -> ")

temp = temp.next

print("None")

dl = DoublyLinkedList()

dl.insert\_beginning(10)

dl.insert\_beginning(20)

dl.insert\_beginning(30)

dl.display()

dl.delete\_beginning()

dl.display()

dl.delete\_end()

dl.display()

OUTPUT:

30 -> 20 -> 10 -> None

20 -> 10 -> None

20 -> None

CIRCULAR LINKEDLIST:  
PROGRAM:  
class Node:

def \_\_init\_\_(self, data):

self.data = data

self.next = None

class CircularLinkedList:

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

self.head = None

def insert\_end(self, data):

new\_node = Node(data)

if self.head is None:

self.head = new\_node

new\_node.next = self.head # Point to itself

else:

temp = self.head

while temp.next != self.head:

temp = temp.next

temp.next = new\_node

new\_node.next = self.head

def display(self):

if self.head is None:

print("List is empty")

return

temp = self.head

while True:

print(temp.data, end=" -> ")

temp = temp.next

if temp == self.head:

break

print("(back to head)")

cll = CircularLinkedList()

cll.insert\_end(10)

cll.insert\_end(20)

cll.insert\_end(30)

cll.display()

OUTPUT:

10 -> 20 -> 30 -> (back to head)