Qno.1)

# Define a Node class to represent nodes in the linked list

class Node:

def \_\_init\_\_(self, val=0, next=None):

self.val = val

self.next = next

# Define a function to delete nodes with a specific value from a linked list

def Delete\_LinkList\_Node(head: Node, key: int) -> Node:

# Create a dummy node to serve as the new head of the linked list

dummy = Node(0)

dummy.next = head

# Initialize two pointers to traverse the linked list

prev = dummy

curr = head

# Traverse the linked list and delete nodes with the specified value

while curr:

if curr.val == key:

prev.next = curr.next

else:

prev = curr

curr = curr.next

# Return the new head of the linked list

return dummy.next

# Create the linked list

head = Node(1, Node(2, Node(1, Node(3))))

# Print the original linked list

curr = head

while curr:

print(curr.val, end="->")

curr = curr.next

print("NULL")

# Delete nodes with a value of 1 from the linked list

head = Delete\_LinkList\_Node(head, 1)

# Print the updated linked list

curr = head

while curr:

print(curr.val, end="->")

curr = curr.next

print("NULL")

Qno.4)

# Define the Node class to create nodes of the linked list

class Node:

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

self.data = data

self.next = None

# Define the LinkedList class to create the linked list

class LinkedList:

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

self.head = None

# Method to insert a node at the end of the linked list

def insert(self, data):

new\_node = Node(data)

if self.head is None:

self.head = new\_node

else:

current\_node = self.head

while current\_node.next is not None:

current\_node = current\_node.next

current\_node.next = new\_node

# Method to reverse the linked list

def reverse(self):

prev\_node = None

current\_node = self.head

while current\_node is not None:

next\_node = current\_node.next

current\_node.next = prev\_node

prev\_node = current\_node

current\_node = next\_node

self.head = prev\_node

# Method to print the linked list

def display(self):

current\_node = self.head

while current\_node is not None:

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

current\_node = current\_node.next

print("NULL")

# Create a linked list with the elements 1, 2, and 3

linked\_list = LinkedList()

linked\_list.insert(1)

linked\_list.insert(2)

linked\_list.insert(3)

# Print the original linked list

print("Original Linked List:")

linked\_list.display()

# Reverse the linked list

linked\_list.reverse()

# Print the reversed linked list

print("Reversed Linked List:")

linked\_list.display()

Qno.5)

# Define the Node class to create nodes of the linked list

class Node:

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

self.data = data

self.next = None

# Define the LinkedList class to create the linked list

class LinkedList:

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

self.head = None

# Method to insert a node at the end of the linked list

def insert(self, data):

new\_node = Node(data)

if self.head is None:

self.head = new\_node

else:

current\_node = self.head

while current\_node.next is not None:

current\_node = current\_node.next

current\_node.next = new\_node

# Method to calculate the average value of the linked list

def average(self):

current\_node = self.head

sum = 0

count = 0

while current\_node is not None:

sum += current\_node.data

count += 1

current\_node = current\_node.next

if count == 0:

return 0

else:

return sum / count

# Method to print the linked list

def display(self):

current\_node = self.head

while current\_node is not None:

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

current\_node = current\_node.next

print("NULL")

# Create a linked list with the elements 1, 2, and 3

linked\_list = LinkedList()

linked\_list.insert(1)

linked\_list.insert(2)

linked\_list.insert(3)

# Print the original linked list

print("Original Linked List:")

linked\_list.display()

# Calculate the average of the linked list

average = linked\_list.average()

# Print the average of the linked list

print("Average:", average)

Qno.3)

# Define the Node class to create nodes of the linked list

class Node:

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

self.data = data

self.next = None

# Define the LinkedList class to create the linked list

class LinkedList:

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

self.head = None

# Method to insert a node at the end of the linked list

def insert(self, data):

new\_node = Node(data)

if self.head is None:

self.head = new\_node

else:

current = self.head

while current.next is not None:

current = current.next

current.next = new\_node

# Method to print the linked list

def print\_list(self):

current = self.head

while current is not None:

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

current = current.next

print("NULL")

# Method to delete duplicated elements from the linked list

def delete\_duplicated\_nodes(self):

if self.head is None:

return

current = self.head

while current is not None:

runner = current

while runner.next is not None:

if runner.next.data == current.data:

runner.next = runner.next.next

else:

runner = runner.next

current = current.next

# Test the program with sample input

if \_\_name\_\_ == '\_\_main\_\_':

linked\_list = LinkedList()

linked\_list.insert(1)

linked\_list.insert(2)

linked\_list.insert(1)

linked\_list.insert(3)

linked\_list.insert(2)

print("Original Linked List:")

linked\_list.print\_list()

linked\_list.delete\_duplicated\_nodes()

print("Linked List after deleting duplicates:")

linked\_list.print\_list()

Qno.2) class Node:

def \_\_init\_\_(self, data=None):

self.data = data

self.next = None

def Delete\_Node\_value(head):

# Create a dictionary to store the count of each unique value

counts = {}

# Traverse the linked list and count the occurrences of each value

current = head

while current is not None:

if current.data not in counts:

counts[current.data] = 0

counts[current.data] += 1

current = current.next

# Create a new linked list with only the unique values

new\_head = None

tail = None

current = head

while current is not None:

if counts[current.data] == 1:

if new\_head is None:

new\_head = Node(current.data)

tail = new\_head

else:

tail.next = Node(current.data)

tail = tail.next

current = current.next

return new\_head

# Define the linked list

head = Node(1)

head.next = Node(2)

head.next.next = Node(1)

head.next.next.next = Node(3)

head.next.next.next.next = Node(2)

# Call the function to delete duplicated values

new\_head = Delete\_Node\_value(head)

# Print the new linked list

current = new\_head

while current is not None:

print(current.data, end=" ")

current = current.next