Data structure Assignment:

Topic: singly Linked List

ARUNTHATHI N

23102009

BTECH .AI & DS

1. Element K present in Singly Linked List or not.

AIM:

To find the element K present in Singly Linked List or not.

ALGORITHM:

Step1:Start

Step2: Definition of a Node in a singly linked list

Step3: Data part of the node

Step4: Constructor to initialize the node with data

Step5: Function to print the linked list

Step6: Printing the above list

Step7:End

PROGRAM:

class Node:

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

self.value=value

self.next=None

class LinkedList:

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

self.head=None

self.tail=None

def Insert\_End(self,val):

NewNode = Node(val)

if self.head is None:

self.head = NewNode

self.tail=self.head

else:

self.tail.next=NewNode

self.tail=NewNode

def Display(self):

temp=self.head

while(temp!=None):

print(temp.value, end='->')

temp=temp.next

Singly= LinkedList( )

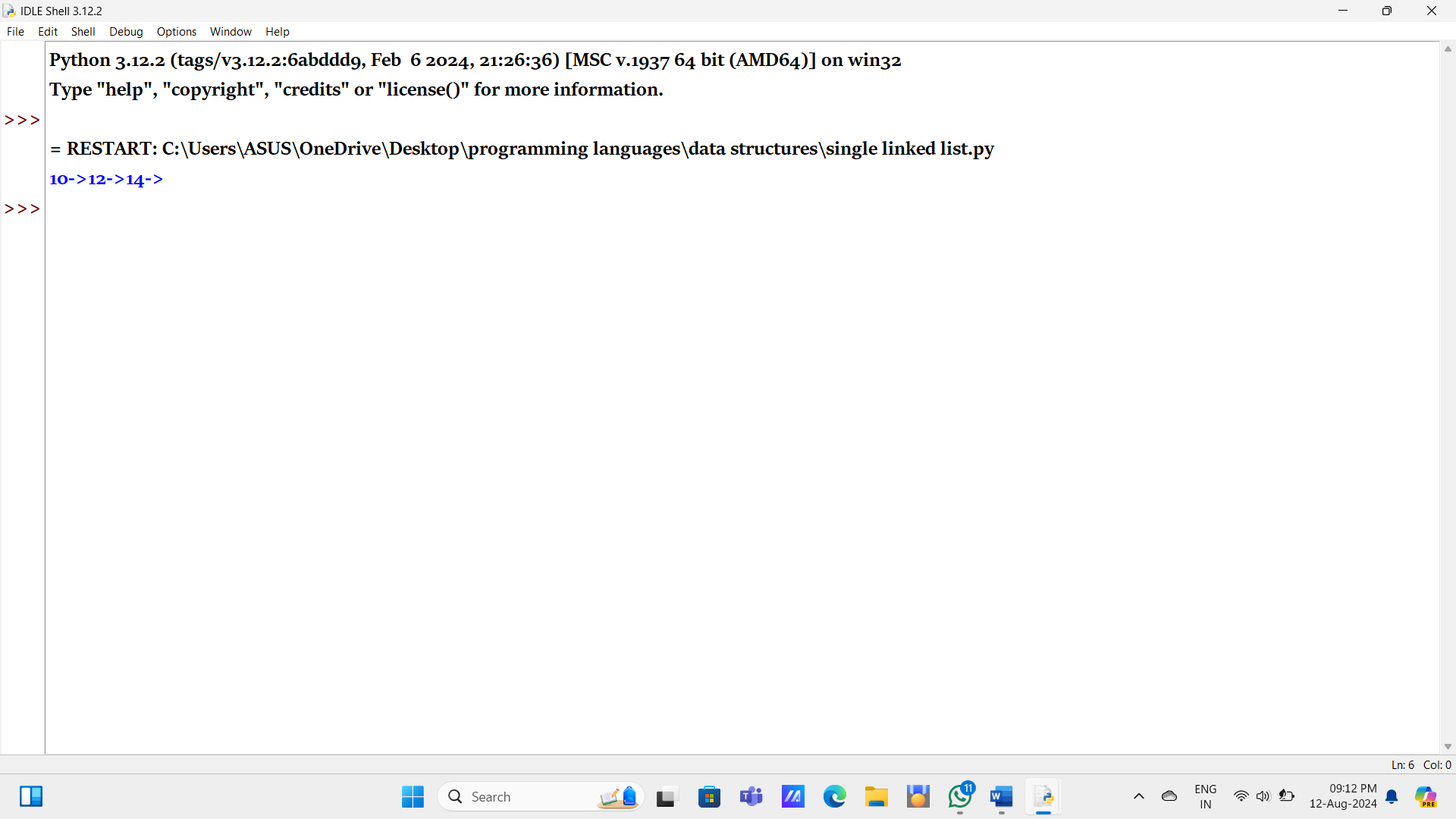
Singly.Insert\_End(10)

Singly.Insert\_End(12)

Singly.Insert\_End(14)

Singly.Display()

OUTPUT:



RESULT:

The element K present in Singly Linked List or not is finded.

2.Implement singly Linked List

AIM:

To implement singly Linked List

ALGORITHM:

Step1:Start

Step2: Move hare K elements ahead

Step3: K is greater than list length

Step4: Move both pointers until hare reaches the end

Step5: Kth to last element

Step6: Printing the above list

Step7:End

PROGRAM:

class Node:

    def \_\_init\_\_(self,data):

        self.data=data

        self.ref=None

class Linked\_list:

    def \_\_init\_\_(self):

        self.head=None

    def printLL(self):

        if self.head is  None:

            print("linked list is empty")

        else:

            n=self.head

            while(n is not None):

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

                n=n.ref

    def add\_begin(self,data):

        newnode=Node(data)

        newnode.ref=self.head

        self.head=newnode

    def add\_end(self,data):

        newnode=Node(data)

        if self.head is None:

            self.head=newnode

        else:

            n=self.head

            while n.ref is not None:

                n=n.ref

            n.ref=newnode

    def after\_add(self,data,x):

        n=self.head

        while n is not None:

            if x==n.data:

                break

            n=n.ref

        if n is None:

            print("linked list is empty")

        else:

            newnode=Node(data)

            newnode.ref=n.ref

            n.ref=newnode

    def before\_add(self,data,x):

        if self.head is None:

            print("Linked list is empty")

            return

        if self.head.data==x:

            newnode=Node(data)

            newnode.ref=self.head

            self.head=newnode

            return

        n=self.head

        while n.ref is not None:

            if n.ref.data==x:

                break

            n=n.ref

        if n.ref is None:

            print("linked list is empty")

        else:

            newnode=Node(data)

            newnode.ref=n.ref

            n.ref=newnode

    def insert\_empty(self,data):

        if self.head is None:

            newnode=Node(data)

            self.head=newnode

        else:

            print("linkedlist is not empty ")

    def delete\_begin(self):

        if self.head is None:

            print("Linked List is empty")

        else:

            self.head=self.head.ref

    def delete\_end(self):

        if self.head is None:

            print("Linkedlist is empty")

        elif(self.head.ref is None):

            self.head=None

        else:

            n=self.head

            while(n.ref.ref is not None):

                n=n.ref

            n.ref=None

    def delete\_by\_value(self,x):

        if self.head is None:

            print("Linkedlist is empty")

            return

        if self.head.data ==x:

            self.head=self.head.ref

            return

        n=self.head

        while n.ref is not None:

            if n.ref.data==x:

                break

            n=n.ref

        if n.ref is None:

                print("That element is not present in the linked list")

        else:

            n.ref=n.ref.ref

LL=Linked\_list()

LL.add\_begin(10)

LL.add\_begin(20)

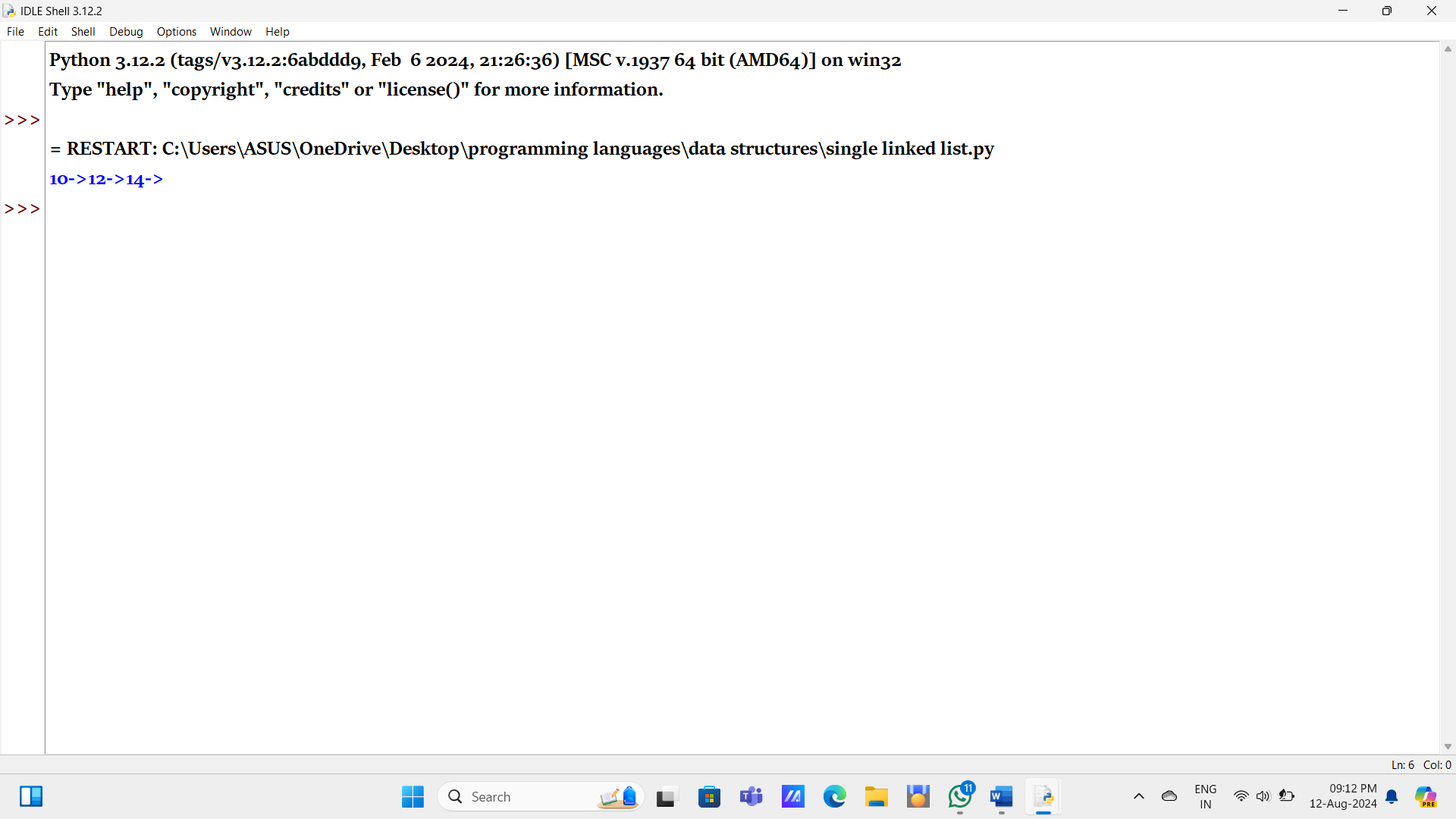
LL.add\_begin(30)

LL.add\_begin(40)

LL.delete\_by\_value(10)

LL.printLL()

OUTPUT:

RESULT:

Implemented singly Linked List.

3.write the program for reverse the singly linked list.

AIM:

To reverse the singly linked list.

ALGORITHM:

Step1:Start

Step2: Reverse current node's pointer

Step3: Update head to the new first node

Step4: Move both pointers until hare reaches the end

Step5: Helper function to print the linked list

Step6: Printing the above list

Step7:End

PROGRAM:

class Node:

    def \_\_init\_\_(self,data):

        self.data=data

        self.ref=None

class Linked\_list:

    def \_\_init\_\_(self):

        self.head=None

    def printLL(self):

        if self.head is  None:

            print("linked list is empty")

        else:

            n=self.head

            while(n is not None):

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

                n=n.ref

    def add\_begin(self,data):

        newnode=Node(data)

        newnode.ref=self.head

        self.head=newnode

def reverse\_print(self):

        nodes=[]

        n=self.head

        while n is not None:

            nodes.append(n.data)

            n=n.ref

        l=len(nodes)-1

        for i in range (l, -1, -1):

            print(nodes[i],end="-->")

LL=Linked\_list()

LL.add\_begin(10)

LL.add\_begin(20)

LL.add\_begin(30)

LL.add\_begin(40)

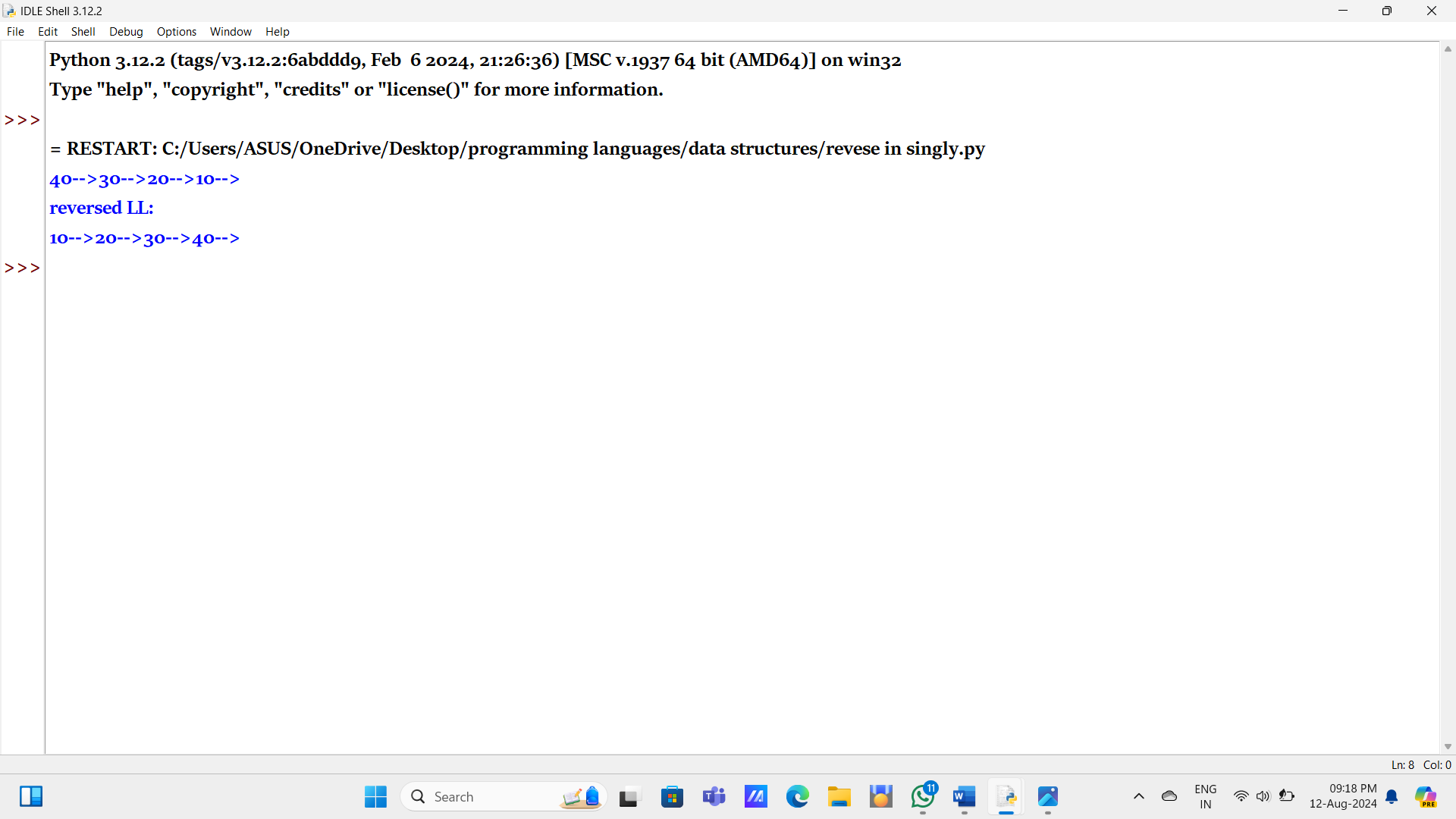
LL.printLL()

print()

print("reversed LL:")

LL.reverse\_print()

OUTPUT:



RESULT:

Printed the singly linked list in reversed order.

4.implementation of circular singly linked list.

AIM:

To implementation of circular singly linked list.

ALGORITHM:

Step1:Start

Step2: Reverse current node's pointer

Step3: Update head to the new first node

Step4: Move both pointers until hare reaches the end

Step5: Helper function to print the linked list

Step6: Printing the above list

Step7:End

PROGRAM:

class Node:

    def \_\_init\_\_(self, data):

        self.data = data

        self.next = None

class CircularLinkedList:

    def \_\_init\_\_(self):

        self.head = None

    def add\_to\_empty(self, data):

        if self.head is not None:

            return

        new\_node = Node(data)

        self.head = new\_node

        self.head.next = self.head

    def add\_to\_begin(self, data):

        if self.head is None:

            self.add\_to\_empty(data)

            return

        new\_node = Node(data)

        new\_node.next = self.head.next

        self.head.next = new\_node

    def add\_to\_end(self, data):

        if self.head is None:

            self.add\_to\_empty(data)

            return

        new\_node = Node(data)

        new\_node.next = self.head.next

        self.head.next = new\_node

        self.head = new\_node

    def traverse(self):

        if self.head is None:

            print("Circular linked list is empty")

            return

        current = self.head.next

        while True:

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

            current = current.next

            if current == self.head.next:

                break

        print()

cll = CircularLinkedList()

cll.add\_to\_empty(10)

cll.add\_to\_begin(20)

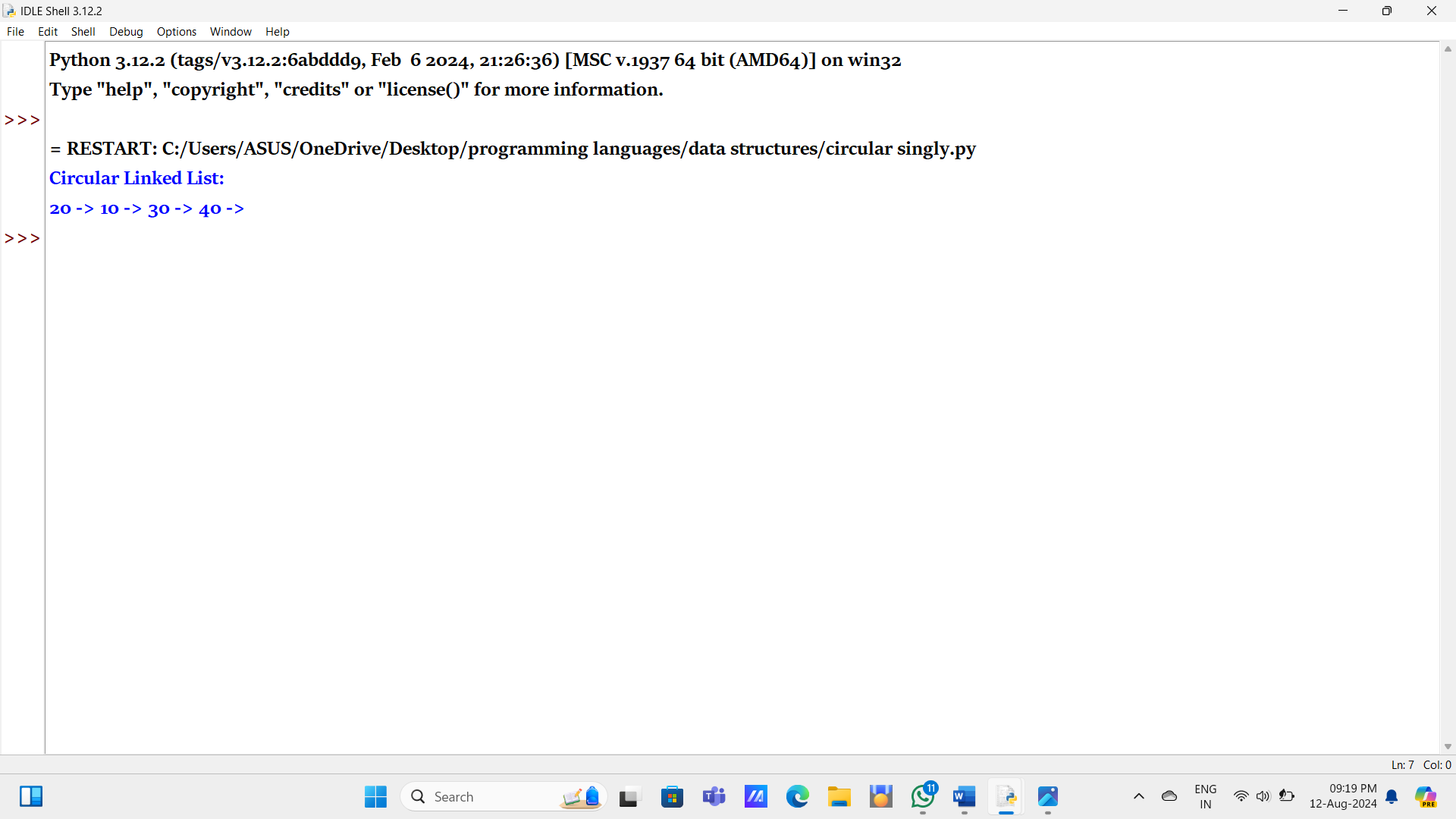
cll.add\_to\_end(30)

cll.add\_to\_end(40)

print("Circular Linked List:")

cll.traverse()

OUTPUT:



RESULT:

To implemented of circular singly linked list.