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

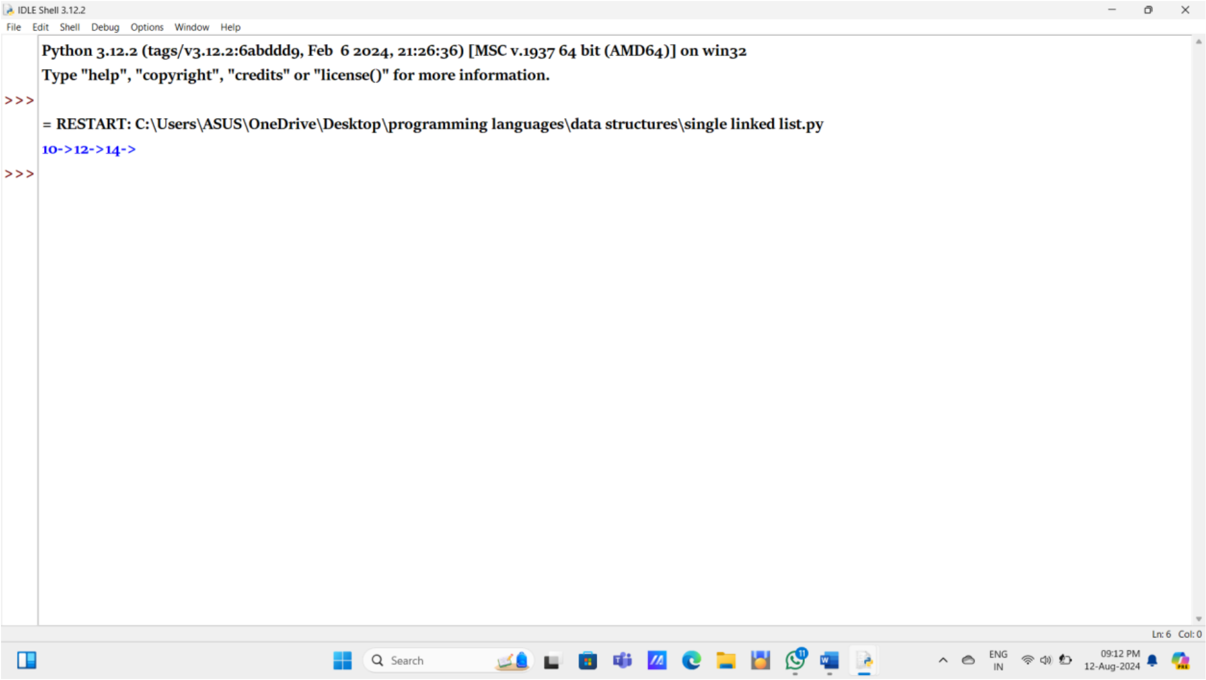
Step6: Printing the above list

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:



```
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\ASUS\OneDrive\Desktop\programming languages\data structures\single linked list.py
10->12->14->
>>>
```

RESULT:

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

2.Implement singly Linked List

AIM:

To implement singly Linked List

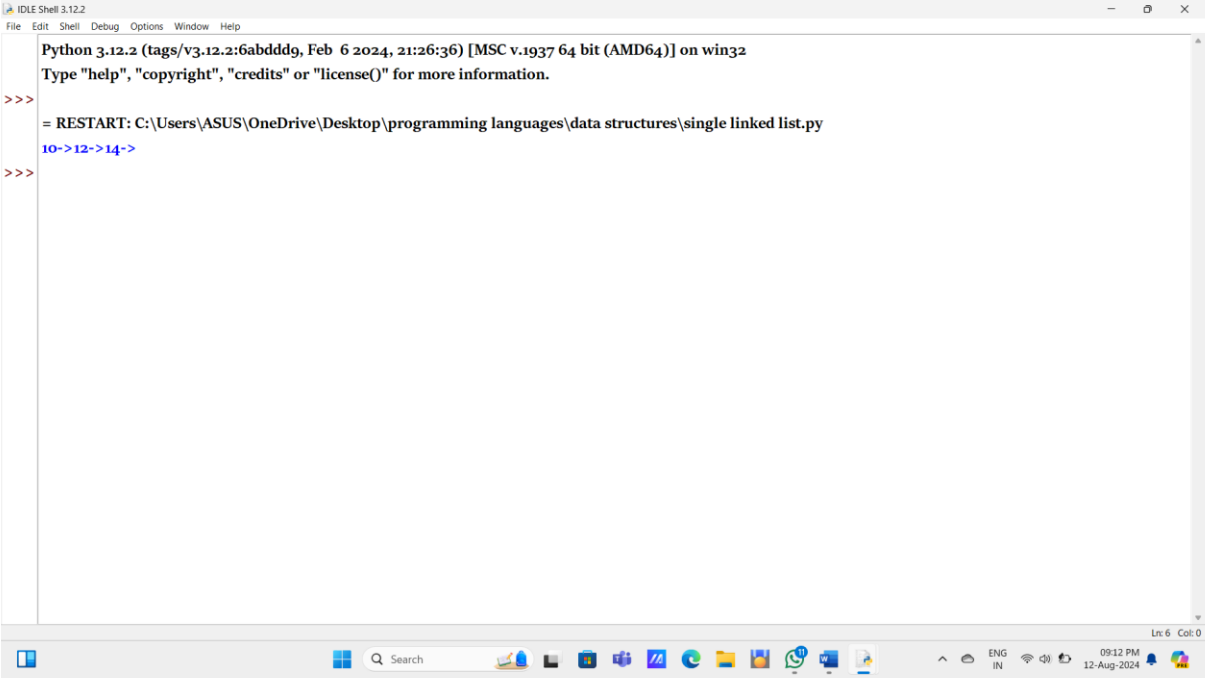
ALGORITHM:

```
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:



```
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\ASUS\OneDrive\Desktop\programming languages\data structures\single linked list.py
10->12->14->
>>>
```

RESULT:

Implemented singly Linked List.

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

AIM:

To reverse the singly linked list.

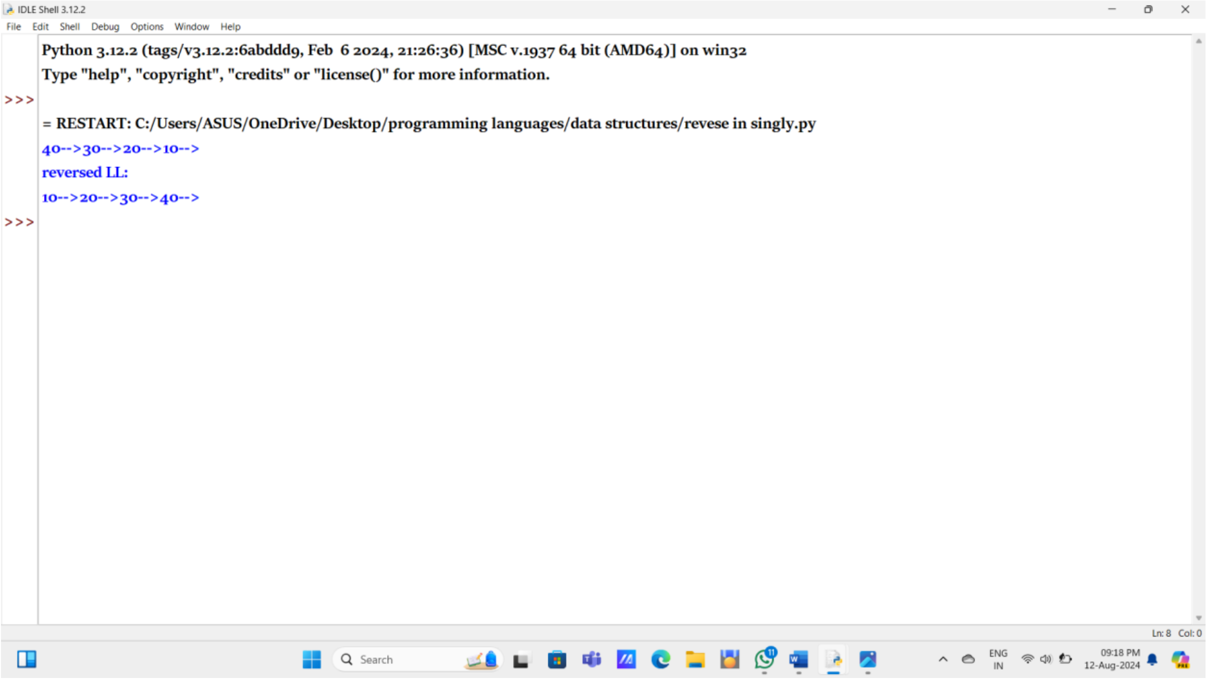
ALGORITHM:

```
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:



```
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/ASUS/OneDrive/Desktop/programming languages/data structures/revese in singly.py
40-->30-->20-->10-->
reversed LL:
10-->20-->30-->40-->
>>>
```

RESULT:

Printed the singly linked list in reversed order.

4.implementation of circular singly linked list.

AIM:

To implementation of circular singly linked list.

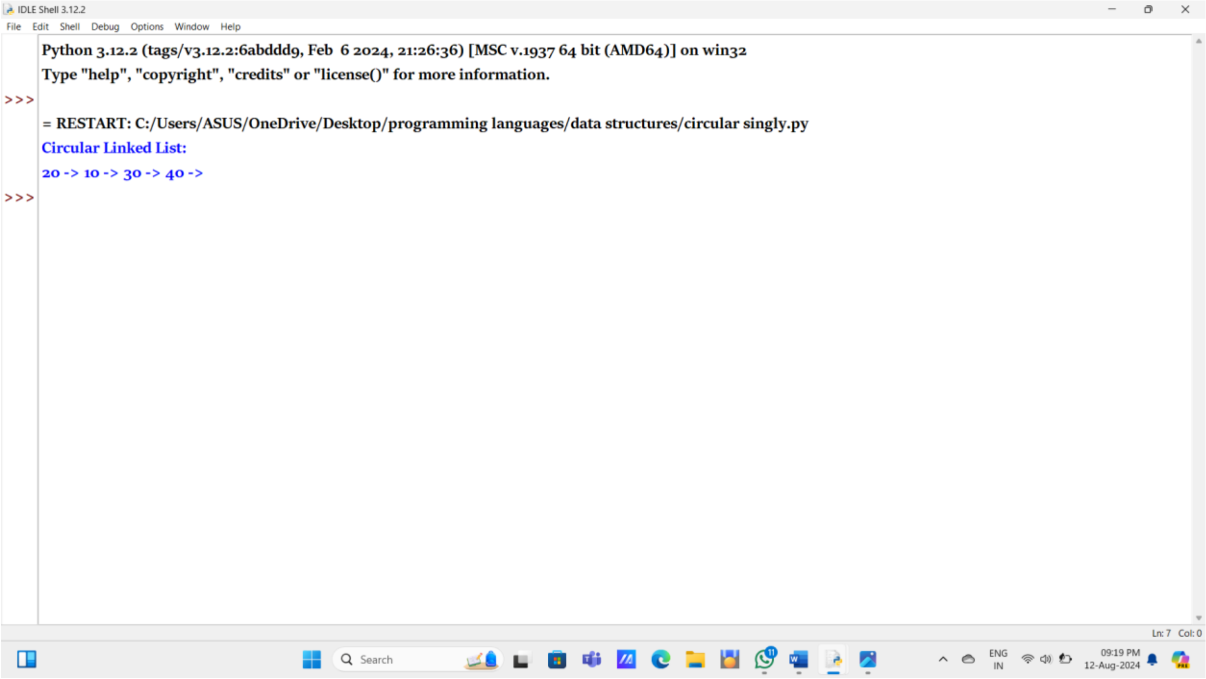
ALGORITHM:

```
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:



```
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/ASUS/OneDrive/Desktop/programming languages/data structures/circular singly.py
Circular Linked List:
20 -> 10 -> 30 -> 40 ->
>>>
```

RESULT:

To implemented of circular singly linked list.