**Data Structures**

**S Abhishek**

**AM.EN.U4CSE19147**

**1. Write a Python program that uses functions to perform the following operations on array.**

a. Creation

b. Insertion (at start, at end, using index, based on value)

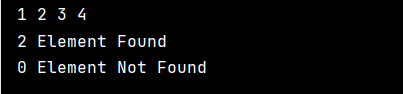
c. Deletion (at start, at end, using index, based on value)

d. Traversal

e. Searching an element. (based on value, based on index)

import array as arr  
  
  
def creation(arr1):  
 arr1.append(**10**)  
 arr1.insert(**0, 0**)  
 arr1.insert(**2, 3**)  
 arr1.insert(**1, 1**)  
  
  
def deletion(arr3):  
 arr3.pop(**0**)  
 arr3.pop()  
 arr3.pop(**5**)  
 arr3.remove(**6**)

def search(arr4):  
 f = **0** for i in arr4:  
 if i == **2**:  
 print("\n2 Element Found")  
 f = **1** break  
 if f == **0**:  
 print("\n2 Element Not Found")  
  
 f = **0** for i in arr4:  
 if i == **0**:  
 print("0 Element Found")  
 f = **1** break  
 if f == **0**:  
 print("0 Element Not Found")  
  
  
def display(arr2): # Traversal  
 for i in range(**0,** len(arr2)):  
 print(arr2[i]**,** end=" ")  
  
  
array = arr.array('i'**,** [**2, 4, 6, 8**])  
  
creation(array) # Creating the Array  
deletion(array) # Deletion of the Elements from Array  
display(array) # Traversal  
search(array) # Searching



**2. Write a Python program that uses functions to perform the following operations on singly linked list.**

a. Creation

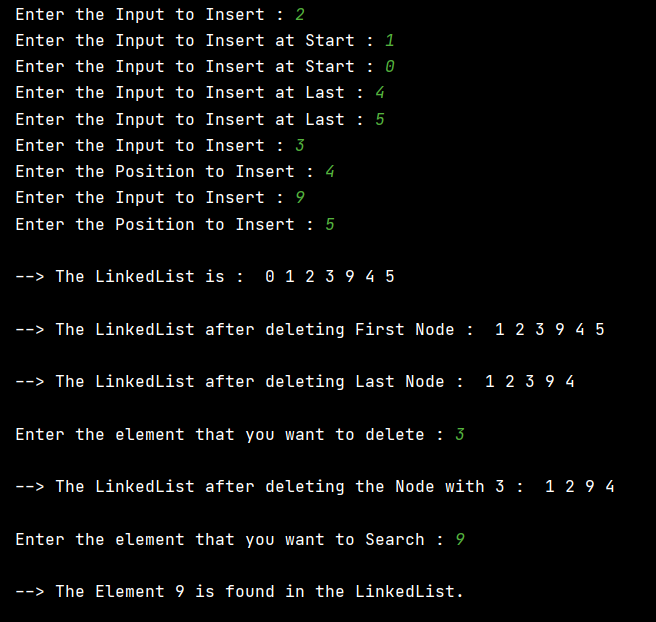
b. Insertion (as first node, as last node, in between node)

c. Deletion (first node, last node, in between node)

d. Traversal

e. Searching an element.

class Node:  
 def \_\_init\_\_(self**,** data):  
 self.data = data  
 self.next = None  
  
  
class LinkedList:  
 def \_\_init\_\_(self):  
 self.head = None  
  
 def print(self):  
 temp = self.head  
 while temp:  
 print(temp.data**,** end=" ")  
 temp = temp.next  
  
 def delete\_first(self):  
  
 temp = self.head  
 self.head = self.head.next  
 temp = None  
  
 def delete\_last(self):  
  
 temp = self.head  
 previous = None  
 while temp.next:  
 previous = temp  
 temp = temp.next  
  
 previous.next = None  
 temp = None  
  
 def delete\_mid(self):  
  
 n = int(input("\n\nEnter the element that you want to delete : "))  
  
 temp = self.head  
 previous = None  
  
 while temp:  
 if temp.data == n:  
 break  
 previous = temp  
 temp = temp.next  
  
 previous.next = temp.next  
 temp = None  
  
 print("\n--> The LinkedList after deleting the Node with {} : ".format(n)**,** end=" ")  
  
 def search(self):  
  
 n = int(input("\n\nEnter the element that you want to Search : "))  
 temp = self.head  
  
 while temp:  
 if temp.data == n:  
 print("\n--> The Element {} is found in the LinkedList.".format(n))  
 break  
 temp = temp.next  
  
 if temp is None:  
 print("\n--> The Element {} is not found in the LinkedList.".format(n))  
  
 def insert\_mid(self):  
  
 # Insertion at Position  
  
 x = int(input("Enter the Input to Insert : "))  
 p = int(input("Enter the Position to Insert : "))  
  
 nod = Node(x)  
 count = **1** temp = self.head  
 while temp:  
 count = count + **1** if count == p:  
 # print(temp.data)  
 nod.next = temp.next  
 temp.next = nod  
 break  
 temp = temp.next  
  
 def insert\_first(self):  
 # Insertion at Start  
  
 x = int(input("Enter the Input to Insert at Start : "))  
 nod = Node(x)  
 temp = self.head  
 nod.next = temp  
 self.head = nod  
  
 def insert\_last(self):  
 # Insertion at Last  
  
 x = int(input("Enter the Input to Insert at Last : "))  
 nod = Node(x)  
 temp = self.head  
 while temp.next:  
 temp = temp.next  
 temp.next = nod  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # --------------------- Creation --------------------------- #  
  
 LL = LinkedList()  
  
 x = int(input("Enter the Input to Insert : "))  
 LL.head = Node(x)  
  
 # --------------------- Insertion --------------------------- #  
  
 # Insertion at Start  
  
 LL.insert\_first()  
 LL.insert\_first()  
  
 #LL.print()  
  
 # Insertion at End  
  
 LL.insert\_last()  
 LL.insert\_last()  
  
 #LL.print()  
  
 # Insertion at Middle  
  
 LL.insert\_mid()  
 #LL.print()  
 LL.insert\_mid()  
  
 # --------------------- Traversal --------------------------- #  
 print("\n--> The LinkedList is : "**,** end=" ")  
 LL.print()  
  
 # --------------------- Deletion --------------------------- #  
  
 # Delete Node 1  
  
 LL.delete\_first()  
  
 print("\n\n--> The LinkedList after deleting First Node : "**,** end=" ")  
 LL.print()  
  
 # Delete Last Node  
  
 LL.delete\_last()  
  
 print("\n\n--> The LinkedList after deleting Last Node : "**,** end=" ")  
 LL.print()  
  
 # Delete InBetween Nodes  
  
 LL.delete\_mid()  
 LL.print()  
  
 # --------------------- Traversal & Searching --------------------------- #  
  
 LL.search()



**3. Write a Python program that uses functions to perform the following operations on doubly linked list**

a. Creation

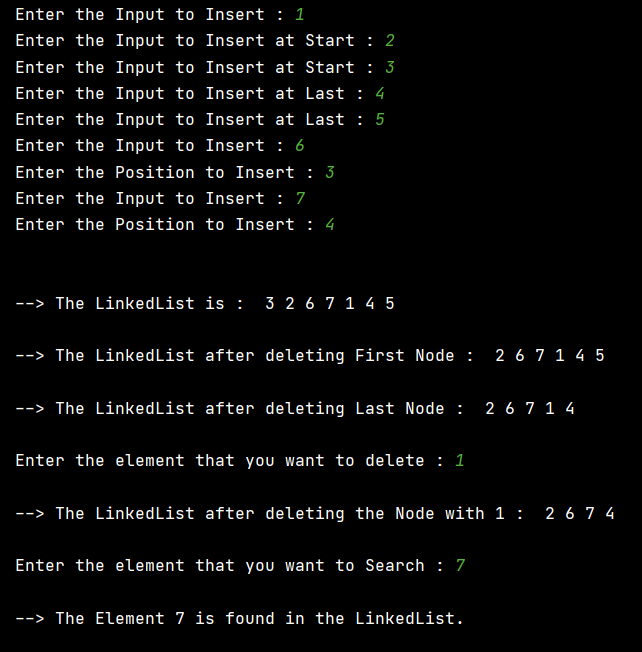
b. Insertion (as first node, as last node, in between node)

c. Deletion (first node, last node, in between node)

d. Traversal

e. Searching an element.

class Node:  
 def \_\_init\_\_(self**,** data):  
 self.previous = None  
 self.data = data  
 self.next = None  
  
  
class LinkedList:  
 def \_\_init\_\_(self):  
 self.head = None  
  
 def print(self):  
 temp = self.head  
 while temp:  
 print(temp.data**,** end=" ")  
 temp = temp.next  
  
 def delete\_first(self):  
  
 temp = self.head  
 self.head = self.head.next  
 # self.head.previous = None  
 temp = None  
  
 def delete\_last(self):  
  
 temp = self.head  
 while temp.next.next:  
 temp = temp.next  
 temp.next = None  
 temp = None  
  
 def delete\_mid(self):  
  
 n = int(input("\n\nEnter the element that you want to delete : "))  
  
 temp = self.head  
  
 while temp:  
 if temp.data == n:  
 break  
 temp = temp.next  
  
 # print(temp.previous.data)  
 # print(temp.data)  
 # print(temp.next.data)  
  
 temp.previous.next = temp.next  
 temp.next.previous = temp.previous  
 temp = None  
  
 print("\n--> The LinkedList after deleting the Node with {} : ".format(n)**,** end=" ")  
  
 def search(self):  
  
 n = int(input("\n\nEnter the element that you want to Search : "))  
 temp = self.head  
  
 while temp:  
 if temp.data == n:  
 print("\n--> The Element {} is found in the LinkedList.".format(n))  
 break  
 temp = temp.next  
  
 if temp is None:  
 print("\n--> The Element {} is not found in the LinkedList.".format(n))  
  
 def insert\_mid(self):  
  
 # Insertion at Position  
  
 x = int(input("Enter the Input to Insert : "))  
 p = int(input("Enter the Position to Insert : "))  
  
 nod = Node(x)  
 count = **0** temp = self.head  
 while temp:  
 count = count + **1** if count == p:  
 # print("\n{}\n".format(temp.data))  
 temp.previous.next = nod  
 nod.previous = temp.previous  
 nod.next = temp  
 temp.previous = nod  
 # print(temp.next.previous.data)  
 break  
 temp = temp.next  
  
 def insert\_first(self):  
 # Insertion at Start  
  
 x = int(input("Enter the Input to Insert at Start : "))  
 nod = Node(x)  
 self.head.previous = nod  
 nod.next = self.head  
 self.head = nod  
  
 def insert\_last(self):  
 # Insertion at Last  
  
 x = int(input("Enter the Input to Insert at Last : "))  
 nod = Node(x)  
 temp = self.head  
 while temp.next:  
 temp = temp.next  
 temp.next = nod  
 nod.previous = temp  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # --------------------- Creation --------------------------- #  
  
 LL = LinkedList()  
  
 x = int(input("Enter the Input to Insert : "))  
 LL.head = Node(x)  
  
 # --------------------- Insertion --------------------------- #  
  
 # Insertion at Start  
  
 LL.insert\_first()  
 LL.insert\_first()  
 LL.print()  
  
 # Insertion at End  
  
 LL.insert\_last()  
 LL.insert\_last()  
  
 LL.print()  
  
 # Insertion at Middle  
  
 LL.insert\_mid()  
 LL.print()  
 LL.insert\_mid()  
  
 # --------------------- Traversal --------------------------- #  
 print("\n\n--> The LinkedList is : "**,** end=" ")  
 LL.print()  
  
 # --------------------- Deletion --------------------------- #  
 # Delete Node 1  
  
 LL.delete\_first()  
  
 print("\n\n--> The LinkedList after deleting First Node : "**,** end=" ")  
 LL.print()  
  
 # Delete Last Node  
  
 LL.delete\_last()  
  
 print("\n\n--> The LinkedList after deleting Last Node : "**,** end=" ")  
 LL.print()  
  
 # Delete InBetween Nodes  
  
 LL.delete\_mid()  
 LL.print()  
  
 # --------------------- Traversal & Searching --------------------------- #  
  
 LL.search()



**4. Write a Python program that uses functions to perform the following operations on circular linked list.**

a. Creation

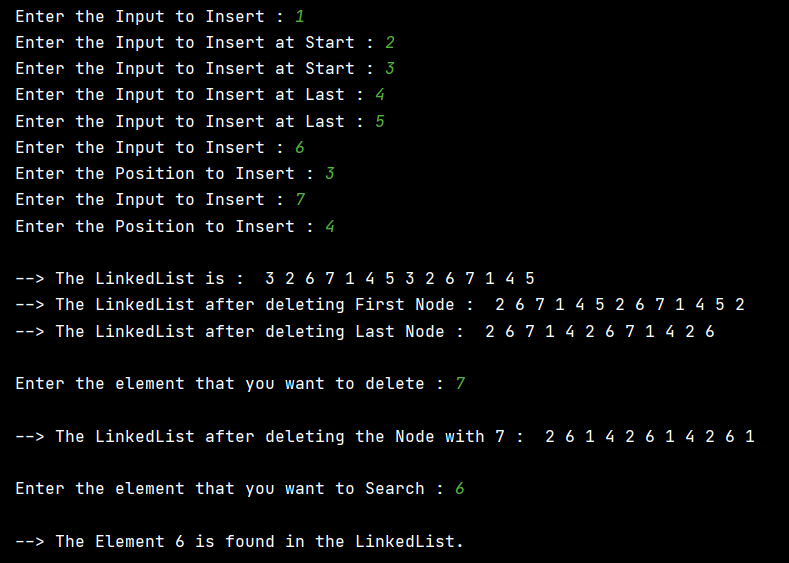
b. Insertion

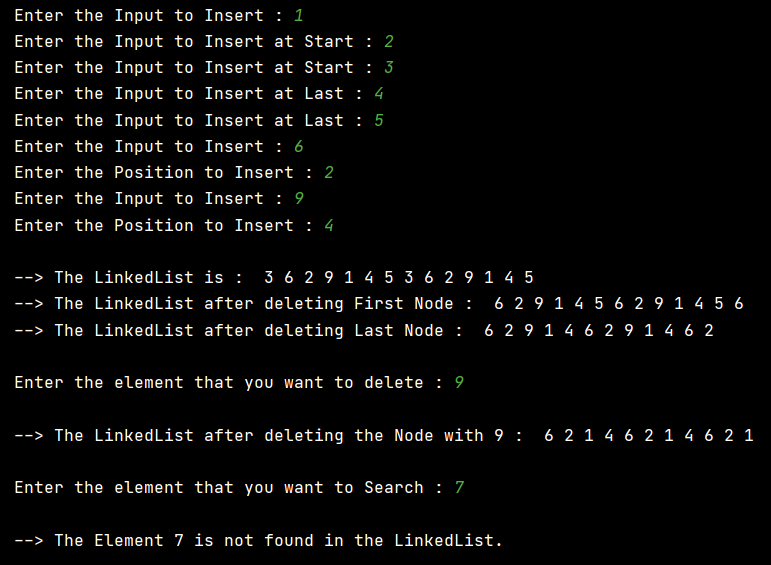
c. Deletion

d. Traversal

e. Searching an element.

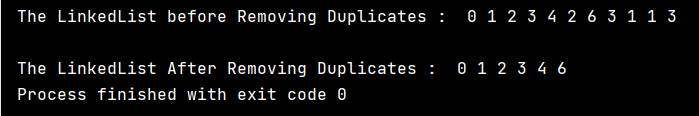
class Node:  
 def \_\_init\_\_(self**,** data):  
 self.previous = None  
 self.data = data  
 self.next = None  
  
  
class LinkedList:  
 def \_\_init\_\_(self):  
 self.head = None  
  
 def print(self):  
 temp = self.head  
 i = **0** global ele\_count  
 while i < ele\_count+**7**:  
 i = i + **1** print(temp.data**,** end=" ")  
 temp = temp.next  
 # print("\n")  
  
 def delete\_first(self):  
  
 self.head = self.head.next  
 # self.head.previous = None  
 i = **1** temp = self.head  
 global ele\_count  
 ele\_count = ele\_count - **1** while i < ele\_count:  
 i = i + **1** temp = temp.next  
 # print("\n")  
 # print(temp.data)  
 self.head.previous = temp  
 temp.next = self.head  
  
 def delete\_last(self):  
  
 temp = self.head  
 global ele\_count  
 ele\_count = ele\_count - **1** i = **0** while i < ele\_count:  
 i = i + **1** temp = temp.next  
  
 temp.previous.next = self.head  
 temp = None  
 #print("\n")  
 #print(temp.data)  
  
 def delete\_mid(self):  
  
 n = int(input("\n\nEnter the element that you want to delete : "))  
  
 temp = self.head  
  
 while temp:  
 if temp.data == n:  
 break  
 temp = temp.next  
  
 # print(temp.previous.data)  
 # print(temp.data)  
 # print(temp.next.data)  
  
 temp.previous.next = temp.next  
 temp.next.previous = temp.previous  
 temp = None  
 global ele\_count  
 ele\_count = ele\_count - **1** print("\n--> The LinkedList after deleting the Node with {} : ".format(n)**,** end=" ")  
  
 def search(self):  
  
 n = int(input("\n\nEnter the element that you want to Search : "))  
 temp = self.head  
  
 i = **1** while i < ele\_count:  
 i = i + **1** if temp.data == n:  
 print("\n--> The Element {} is found in the LinkedList.".format(n))  
 break  
 temp = temp.next  
  
 if i == ele\_count:  
 print("\n--> The Element {} is not found in the LinkedList.".format(n))  
  
 def insert\_mid(self):  
  
 # Insertion at Position  
  
 x = int(input("Enter the Input to Insert : "))  
 p = int(input("Enter the Position to Insert : "))  
  
 nod = Node(x)  
 count = **0** temp = self.head  
 while temp:  
 count = count + **1** if count == p:  
 # print("\n{}\n".format(temp.data))  
 temp.previous.next = nod  
 nod.previous = temp.previous  
 nod.next = temp  
 temp.previous = nod  
 # print(temp.next.previous.data)  
 break  
 temp = temp.next  
 global ele\_count  
 ele\_count = ele\_count + **1** def insert\_first(self):  
 # Insertion at Start  
  
 x = int(input("Enter the Input to Insert at Start : "))  
 nod = Node(x)  
 self.head.previous = nod  
 nod.next = self.head  
 self.head = nod  
 temp = self.head  
 i = **0** global ele\_count  
 while i < ele\_count:  
 i = i + **1** temp = temp.next  
  
 temp.next = self.head  
 self.head.previous = temp  
  
 ele\_count = ele\_count + **1** def insert\_last(self):  
 # Insertion at Last  
  
 x = int(input("Enter the Input to Insert at Last : "))  
 nod = Node(x)  
 temp = self.head  
 global ele\_count  
 i = **1** while i < ele\_count:  
 i = i + **1** temp = temp.next  
  
 # print(temp.data)  
 temp.next = nod  
 nod.previous = temp  
 self.head.previous = nod  
 nod.next = self.head  
 ele\_count = ele\_count + **1**if \_\_name\_\_ == '\_\_main\_\_':  
 # --------------------- Creation --------------------------- #  
  
 LL = LinkedList()  
  
 x = int(input("Enter the Input to Insert : "))  
 LL.head = Node(x)  
  
 global ele\_count  
 ele\_count = **1** # --------------------- Insertion --------------------------- #  
  
 # Insertion at Start  
  
 LL.insert\_first()  
 LL.insert\_first()  
  
 # LL.print()  
  
 # Insertion at End  
  
 LL.insert\_last()  
 LL.insert\_last()  
  
 # LL.print()  
  
 # Insertion at Middle  
  
 LL.insert\_mid()  
 # LL.print()  
 LL.insert\_mid()  
 #LL.print()  
  
 # print("\n")  
 # print(ele\_count)  
  
 # --------------------- Traversal --------------------------- #  
  
 print("\n--> The LinkedList is : "**,** end=" ")  
 LL.print()  
  
 # --------------------- Deletion --------------------------- #  
  
 # Delete Node 1  
  
 LL.delete\_first()  
  
 print("\n--> The LinkedList after deleting First Node : "**,** end=" ")  
 LL.print()  
  
 # Delete Last Node  
  
 LL.delete\_last()  
  
 print("\n--> The LinkedList after deleting Last Node : "**,** end=" ")  
 LL.print()  
  
 # Delete InBetween Nodes  
  
 LL.delete\_mid()  
  
 LL.print()  
  
 # print("\n")  
 # print(ele\_count)  
 # --------------------- Traversal & Searching --------------------------- #  
  
 LL.search()





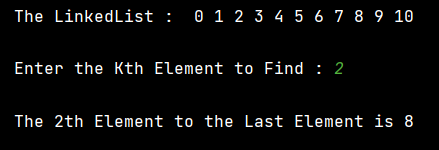
**5. Write a Python program to remove duplicates from an unsorted linked list.**

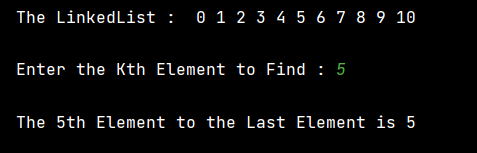
class Node:  
 def \_\_init\_\_(self**,**data):  
 self.previous = None  
 self.data = data  
 self.next = None  
  
class LinkedList:  
 def \_\_init\_\_(self):  
 self.head = None  
  
 def print(self):  
 temp = self.head  
 while temp:  
 print(temp.data**,**end=" ")  
 temp = temp.next  
  
 def dup(self):  
  
 present\_node = next\_node = self.head  
 while present\_node is not None:  
 while next\_node.next is not None:  
 if next\_node.next.data == present\_node.data:  
 next\_node.next = next\_node.next.next  
 else:  
 next\_node = next\_node.next  
 present\_node = next\_node = present\_node.next  
  
if \_\_name\_\_=='\_\_main\_\_':  
  
 LL = LinkedList()  
 nod0 = Node(**0**)  
 LL.head = nod0  
  
 nod1 = Node(**1**)  
 nod0.next = nod1  
  
 nod2 = Node(**2**)  
 nod1.next = nod2  
  
 nod3 = Node(**3**)  
 nod2.next = nod3  
  
 nod4= Node(**4**)  
 nod3.next = nod4  
  
 nod5 = Node(**2**)  
 nod4.next = nod5  
  
 nod6 = Node(**6**)  
 nod5.next = nod6  
  
 nod7 = Node(**3**)  
 nod6.next = nod7  
  
 nod8 = Node(**1**)  
 nod7.next = nod8  
  
 nod9 = Node(**1**)  
 nod8.next = nod9  
  
 nod10 = Node(**3**)  
 nod9.next = nod10  
  
 print("The LinkedList before Removing Duplicates : "**,**end=" ")  
 LL.print()  
  
 LL.dup()  
 print("\n\nThe LinkedList After Removing Duplicates : "**,** end=" ")  
 LL.print()

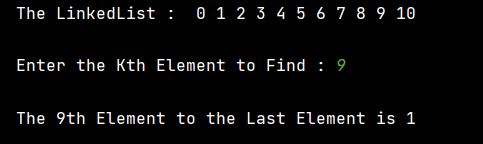
****

**6. Write a Python program to implement an algorithm to find the k’th to the last element of a singly linked list.`**

class Node:  
 def \_\_init\_\_(self**,**data):  
 self.previous = None  
 self.data = data  
 self.next = None  
  
class LinkedList:  
 def \_\_init\_\_(self):  
 self.head = None  
  
 def print(self):  
 temp = self.head  
 while temp:  
 print(temp.data**,**end=" ")  
 temp = temp.next  
  
 def find(self):  
  
 n = int(input("\n\nEnter the Kth Element to Find : "))  
  
 count = **0** temp = self.head  
 while temp:  
 count = count + **1** temp = temp.next  
  
 count = count - n  
 temp = self.head  
 while count>**1**:  
 count = count - **1** temp = temp.next  
  
 print("\nThe {}th Element to the Last Element is {} ".format(n**,**temp.data))  
  
  
if \_\_name\_\_=='\_\_main\_\_':  
  
 LL = LinkedList()  
 nod0 = Node(**0**)  
 LL.head = nod0  
  
 nod1 = Node(**1**)  
 nod0.next = nod1  
  
 nod2 = Node(**2**)  
 nod1.next = nod2  
  
 nod3 = Node(**3**)  
 nod2.next = nod3  
  
 nod4= Node(**4**)  
 nod3.next = nod4  
  
 nod5 = Node(**5**)  
 nod4.next = nod5  
  
 nod6 = Node(**6**)  
 nod5.next = nod6  
  
 nod7 = Node(**7**)  
 nod6.next = nod7  
  
 nod8 = Node(**8**)  
 nod7.next = nod8  
  
 nod9 = Node(**9**)  
 nod8.next = nod9  
  
 nod10 = Node(**10**)  
 nod9.next = nod10  
  
 print("The LinkedList : "**,**end=" ")  
  
 LL.print()  
  
 LL.find()

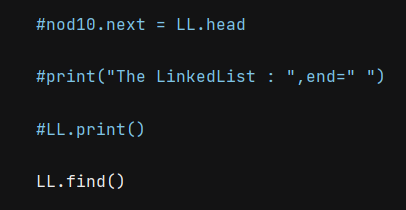
****

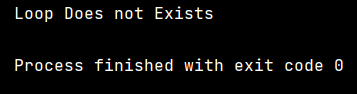
****

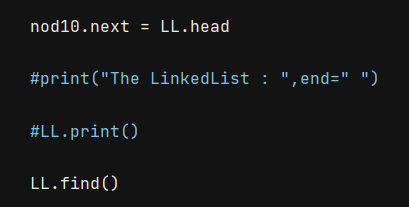
****

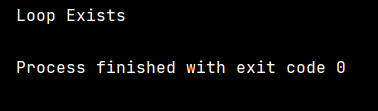
**Write a Python program to detect if a linked list has a loop in it.**

class Node:  
 def \_\_init\_\_(self**,**data):  
 self.data = data  
 self.next = None  
 self.flag = **0**class LinkedList:  
 def \_\_init\_\_(self):  
 self.head = None  
  
 def print(self):  
 temp = self.head  
 while temp:  
 print(temp.data**,**end=" ")  
 temp = temp.next  
  
 def find(self):  
 temp = self.head  
 while temp:  
 if temp.flag == **1**:  
 print("Loop Exists")  
 return  
 temp.flag = **1** temp = temp.next  
 print("Loop Does not Exists")  
  
if \_\_name\_\_=='\_\_main\_\_':  
  
 LL = LinkedList()  
 nod0 = Node(**0**)  
 LL.head = nod0  
  
 nod1 = Node(**1**)  
 nod0.next = nod1  
  
 nod2 = Node(**2**)  
 nod1.next = nod2  
  
 nod3 = Node(**3**)  
 nod2.next = nod3  
  
 nod4= Node(**4**)  
 nod3.next = nod4  
  
 nod5 = Node(**5**)  
 nod4.next = nod5  
  
 nod6 = Node(**6**)  
 nod5.next = nod6  
  
 nod7 = Node(**7**)  
 nod6.next = nod7  
  
 nod8 = Node(**8**)  
 nod7.next = nod8  
  
 nod9 = Node(**9**)  
 nod8.next = nod9  
  
 nod10 = Node(**10**)  
 nod9.next = nod10  
  
 #nod10.next = LL.head  
  
 #print("The LinkedList : ",end=" ")  
  
 #LL.print()  
  
 LL.find()

****

****

****

****

**Thankyou !!**