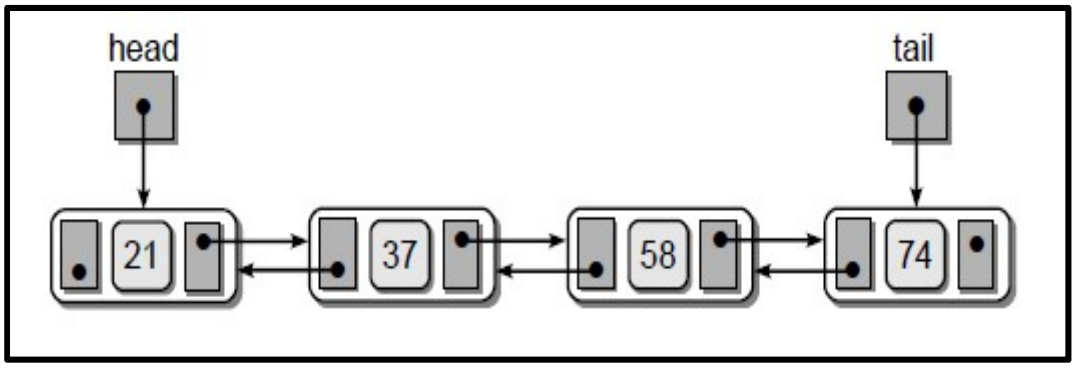
Week 7	Doubly Linked Lists, Examples: Image Viewer, Music Player etc., Applications. DLL Node, List operations - Create, Appending nodes, Delete, Search. Circular Doubly Linked Lists – Organization, List operations – Appending Nodes, delete, traversal.

Doubly Linked List

- The singly linked list consists of nodes with data field and single link field.
- Access and traversals starts with the first node and moves towards the last node, one node at a time.
- But what if we want to traverse the nodes in reverse order?
- With the singly linked list, this can be done but not efficiently.
- For these kind of operations and problems, we need direct access to both the immediate previous node and next node for a given node.

Doubly Linked List

- In a doubly linked list, each node contains data field and two link fields (One link refers previous node and other link refers next node).
- A head reference is used to reference the first node in the list.
- A tail reference is used to reference the last node in the list.
- **Example:**



- **There are several operations that are commonly performed on a doubly linked list:**
  - **Creating Nodes**
  - **Traversal of Nodes**
  - **Searching for a Node**
  - **appending Nodes**
  - **Removing Nodes**

Creating Nodes

- We can create a Node by defining basic class containing data field and two link fields.
- Data field contains some value and one link field contains refersprevious node and other link refers next node.

```
class Node:
    def __init__(self,data):
        self.data = data
        self.prev = None
        self.next = None
```

Traversing the Nodes

- It means, accessing each and every node of the Linked List in sequence.
- The process starts by assigning a temporary variable **curNode** to point to the first/Head node of the list.
- After entering the loop, the value stored in the first node is printed by accessing the data field the node using the **curNode** variable.
- The **curNode** variable is then advanced to the next node by assigning the current node's next field
- The loop continues until every node in the linked list has been accessed (till **CurNode** variable becomes None).
- The end of the Linked List/Traversal is identified when **curNode** becomes None/Null.

```
def traversal(self):
    curNode = self.head
        while curNode is not None:
    print(curNode.data)
    curNode = curNode.next
```

Searching for a Node

- A linear search operation can be performed on a linked list.
- It is very similar to the traversal operation. The only difference is that the loop ends early if we found the key value within the list.
- There are two conditions in the while loop.
- First, check whether curNode is None/Null or not. If it is not None, then check data field of curNode with the search key.

- If the Key is found in the list, curNode will not be None
- If the Key is not found in the list, curNode will be None since the end of the listis reached.

```
def  serach(self, key):
        curNode = self.head
        while curNode is not None and curNode.data != key:
            curNode = curNode.next
        return curNode is not None
```

Appending Nodes

- In Linked List. We can insert/add nodes at any location/position:
  - We can insert node at the beginning of the linked list (Prepending Node)
  - We can insert node at the end of the linked list (Appending Node)
  - We can insert node at the specific location/position of the linked list.
- Appending a node can be done as follows:
  - First, we must create a new node to store the new value.
  - Then set temporary **curNode** variable to point to the node currently at the front/head node of the list.
  - We then move to the next node using the next filed of **curNode** and makes it **curNode**. It will be repeated till end of the list is reached (till link field of **curNode** becomes None).
  - Now, Change the next filed of **curNode** to point or refer new Node and change the prev field of the new node to curNode.

```
def append(self,value):
        newNode = Node(value)
        curNode = self.head
        while curNode.next is not None:
            curNode = curNode.next
        curNode.next= newNode
        newNode.prev = curNode
        self.tail=newNode
```

Removing/Deleting Nodes

- An item can be removed from a linked list by removing or unlinking the node containing that item.
- First, we must find the node containing the target.
- After finding the node, it has to be unlinked from the list, which means changing the next field of the node's predecessor to point to its successor and changing the prev field of the successor's node to point to its predecessor.

```
def  remove(self,target):
        curNode = self.head
        while curNode is not None and curNode.data !=target:
                curNode = curNode.next
                if curNode is not None:
                        if curNode is self.head:
                                curNode.next.prev=None
                                self.head=curNode.next
                        else:
                                curNode.prev.next = curNode.next
                                curNode.next.prev = curNode.prev
                else:
                        print("\n",target,"not found in the list")
```

Program #1: Python Program to implement Doubly Linked List

```
class Node:
    def __init__(self,data):
        self.data = data
        self.prev = None
        self.next = None

class LinkedList:
   def __init__(self,value):
        self.head = Node(value)
        self.tail = self.head
```

```
    def append(self,value):
        newNode = Node(value)
        curNode = self.head
        while curNode.next is not None:
            curNode = curNode.next
        curNode.next= newNode
        newNode.prev = curNode
        self.tail=newNode
        print("Node",value,"appended.")
    def traversal(self):
        curNode = self.head
        if curNode is None:
            print("Doubly Linked List is empty.")
        else:
            print("Contents of Doubly Linked List are")
            while curNode is not None:
                print(curNode.data,end="\t")
                curNode = curNode.next
    def remove(self,target):
        curNode = self.head
        while curNode is not None and curNode.data !=target:
            curNode = curNode.next
        if curNode is not None:
            if curNode is self.head:
                curNode.next.prev=None
                self.head=curNode.next
            elif curNode is self.tail:
                curNode.prev.next=None
                self.tail=curNode.prev
            else:
                curNode.prev.next = curNode.next
                curNode.next.prev = curNode.prev
            print("Node",target,"deleted.")
        else:
            print(target,"not found in the list")
    def search(self,target):
        curNode = self.head
        while curNode is not None:
            if curNode.data == target:
                return True
            else:
                curNode = curNode.next
        return False
```

```
LL = LinkedList(10)
LL.traversal()
LL.append(20)
LL.append(30)
LL.traversal()
print("Node 40 exits?:",LL.search(40))
print("Node 20 exits?:",LL.search(20))
print("Node 10 exits?:",LL.search(30))
LL.remove(10)
LL.traversal()
LL.remove(30)
LL.traversal()
LL.remove(50)
LL.traversal()
```

Output #1:

Contents of Doubly Linked List are

10

Node 20 appended.

Node 30 appended.

Contents of Doubly Linked List are

10 20 30

Node 40 exits?: False

Node 20 exits?: True

Node 10 exits?: True

Node 10 deleted.

Contents of Doubly Linked List are

20 30

Node 30 deleted.

Contents of Doubly Linked List are

20

50 not found in the list

Contents of Doubly Linked List are

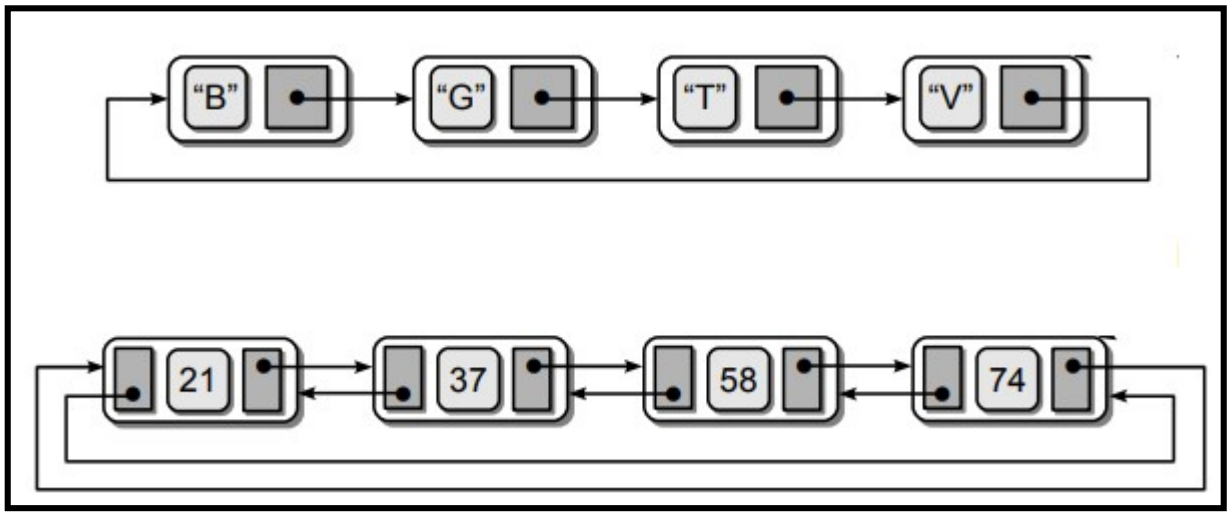
20

Circular Doubly Linked List

- In a Circular doubly linked list, each node contains data field and two link fields (next and prev, prev link refers previous node and next link refers next node).
- A head reference is used to reference the first node in the list.
- A tail reference is used to reference the last node in the list.

Organization

- **next link field of last node refers first node**
- **prev link field of first node refers last node**
- **Example:**



- **There are several operations that are commonly performed on a circular doubly linked list:**
  - **Traversal of Nodes**
  - **appending Nodes**
  - **Removing Nodes**

Traversing the Nodes

- It means accessing each and every node of the Linked List in sequence.
- The process starts by assigning a temporary variable **curNode** to point to the first/Head node of the list.
- After entering the loop, if the curNode is not a tail node, then the value stored in the curNode is printed by accessing the data field the node.

- The **curNode** variable is then advanced to the next node by assigning the current node's next field.
- The loop continues until every node in the linked list has been accessed (till **CurNode** is the tail node).
- Finally, display the value stored in the tail node.

```
def traversal(self):
    curNode = self.head
    while curNode is not self.tail:
        print(curNode.data,end="\t")
        curNode = curNode.next
    print(curNode.data)
```

Appending Nodes

- In Linked List. We can insert/add nodes at any location/position:
  - We can insert node at the beginning of the linked list (Prepending Node)
  - We can insert node at the end of the linked list (Appending Node)
  - We can insert node at the specific location/position of the linked list.
- Appending a node can be done as follows:
  - First, we must create a new node to store the new value.
  - Assign a temporary variable **curNode** to point to the first/Head node of the list.
  - After entering the loop, if the curNode is not a tail node, then the value stored in the curNode is printed by accessing the data field the node.
  - The **curNode** variable is then advanced to the next node by assigning the current node's next field.
  - The loop continues until every node in the linked list has been accessed (till **CurNode** is the tail node).
  - Finally, set next field of curNode to new Node and previous link field of new node to curNode.
  - Set next filed of new node to head node and previous link field of head node to new node.

```
def append(self,value):
        newNode = Node(value)
        curNode = self.head
        while curNode is not self.tail:
            curNode = curNode.next
        curNode.next= newNode
         newNode.prev = curNode
         newNode.next = self.head
         self.head.prev=newNode
        self.tail=newNode
```

Removing/Deleting Nodes

- An item can be removed from a linked list by removing or unlinking the node containing that item.
- First, we must find the node containing the target.
- After finding the node, it has to be unlinked from the list, which means changing the next field of the node's predecessor to point to its successor and changing the prev field of the successor's node to point to its predecessor.

```
def  remove(self,target):
        curNode = self.head
        while curNode is not None and curNode.data !=target:
            curNode = curNode.next
            if curNode is not None:
                if curNode is self.head:
                    curNode.next.prev=None
                    self.head=curNode.next
                else:
                    curNode.prev.next = curNode.next
                    curNode.next.prev = curNode.prev
            else:
                print("\n",target,"not found in the list")
```

Program #2: Python Program to implement Circular Doubly Linked List

```
class Node:
    def __init__(self,data):
        self.data = data
        self.prev = None
        self.next = None

class LinkedList:
    def __init__(self,value):
        self.head = Node(value)
        self.tail=self.head
        self.head.next=self.head
        self.head.prev=self.head

    def append(self,value):
        newNode = Node(value)
        curNode = self.head
        while curNode is not self.tail:
            curNode = curNode.next
        curNode.next= newNode
        newNode.prev = curNode
        newNode.next = self.head
        self.head.prev=newNode
        self.tail=newNode
        print("Node",value,"appended")

    def traversal(self):
        curNode = self.head
        if curNode is None:
            print("Circular Doubly Linked List is empty")
        else:
            print("Contents of Circular Doubly Linked List are")
            while curNode is not self.tail:
                print(curNode.data,end="\t")
                curNode = curNode.next
            print(curNode.data)
```

```
    def remove(self,target):
        curNode = self.head
        while curNode is not self.tail and curNode.data !=target:
            curNode = curNode.next
        if curNode is not self.tail:
            if curNode is self.head:
                curNode.next.prev=self.tail
                self.head=curNode.next
                self.tail.next=self.head
            else:
                curNode.prev.next = curNode.next
                curNode.next.prev = curNode.prev
            print("Node",target,"deleted")
        elif curNode is self.tail:
            self.tail.prev.next=self.head
            self.head.prev=self.tail.prev
            self.tail=self.tail.prev
            print("Node",target,"deleted")
        else:
            print(target,"not found in the list")


LL = LinkedList(10)

LL.traversal()

LL.append(20)
LL.append(30)
LL.traversal()

LL.remove(20)
LL.traversal()

LL.remove(30)
LL.traversal()
```

Output #1:

Contents of Circular Doubly Linked List are

10

Node 20 appended

Node 30 appended

Contents of Circular Doubly Linked List are

10 20 30

Node 20 deleted

Contents of Circular Doubly Linked List are

10 30

Node 30 deleted

Contents of Circular Doubly Linked List are

10

Activity #7

1. **Develop python function for following operations on doubly linked list:**
    a. Traversal of doubly linked list in reverse order
    b. Find the length of the list
    c. Search for an item
    d. Prepending Node

2. **Develop Python Program to perform following operations on doubly linked list:**
    a. Insert 10, 30, 50, 60
    b. Display List
    c. Insert 90
    d. Display List
    e. Remove 50
    f. Display List
    g. Remove 30
    h. Insert 80
    i. Display List

3. **Develop Python Program to perform following operations on circular doubly linked list:**
    a. Insert 10, 30, 50, 60
    b. Display List
    c. Insert 90
    d. Display List
    e. Remove 50
    f. Display List
    g. Remove 30
    h. Insert 80
    i. Display List