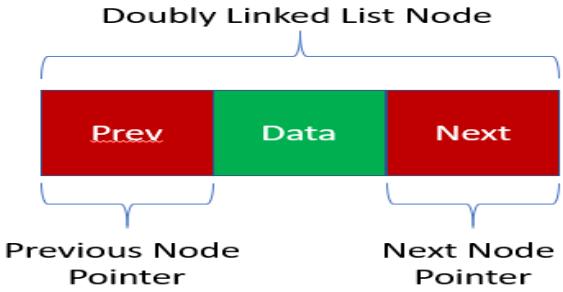
Linked Lists

linked list is a linear data structure where elements, called **nodes**, are connected using pointers. Unlike arrays, linked lists do not require contiguous memory locations, making them flexible for dynamic memory allocation.
NOTE: Nodes are scattered in the memory, they do not stored in contiguous memory locations like array

Types of Linked Lists

1. **Singly Linked List**:
    a. Each node has two parts:
        i. **Data**: The value stored in the node. ii. **Next**: A pointer/reference (memory address) to the next node.
        iii. 

    b. Traversal is one-directional (from head to tail).
2. **Doubly Linked List**:
    a. Each node has three parts:
        i. **Data**: The value stored in the node.
        ii. **Next**: A pointer/reference to the next node.
        iii. **Prev**: A pointer/reference to the previous node.
        iv. 

    b. Traversal is bidirectional (both forward and backward).

3. **Circular Linked List**:
    a. The last node points back to the first node, forming a circle.
    b. Can be singly or doubly linked.

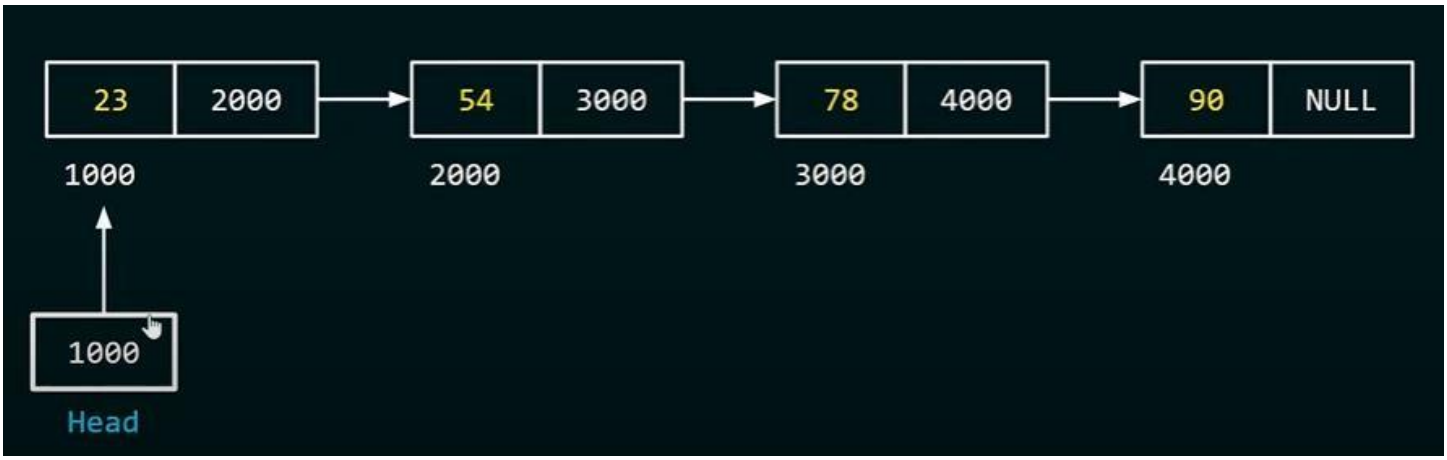
Advantages of Linked Lists

- **Dynamic Size**: Can grow or shrink as needed.
- **Efficient Insertions/Deletions**: Adding/removing elements is efficient if the pointer to the target node is known.
- **Memory Utilization**: No need for a contiguous block of memory.

Disadvantages of Linked Lists

- **Sequential Access**: Accessing elements requires traversal from the head, making it slower than arrays for random access.
- **Memory Overhead**: Requires extra memory for pointers.

Singly Linked List



Basic Example:

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

# creating nodes
node1 = Node(10)
node2 = Node(20)
node3 = Node(30)
node4 = Node(40)

# connecting nodes
node1.next = node2
node2.next = node3
node3.next = node4

# printing linked nodes
current = node1
while current is not None:
    print(current.data,end='->')
    current = current.next
print('None')
OutPut: 10->20->30->40->None
```

Advance Example:

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

class LinkedList:
    def __init__(self):
        self.head = None

    def add_to_end(self,data):
        new_node = Node(data)

        # for first element
        if not self.head:
            # made first element the head of linked list
            self.head = new_node
            return
        current = self.head
        # when current node has its NEXT node connected
        #  make the next node as current
        while current.next:
            current = current.next
        # making new node as the NEXT of current node
        current.next = new_node

    def display(self):
        current = self.head
        while current is not None:
            print(current.data,end=' -> ')
            current = current.next
        print('None')

l1 = LinkedList()
l1.add_to_end(4)
l1.add_to_end(6)
l1.add_to_end(8)
l1.add_to_end(1)
l1.display()

# Output: 4->6->8->1->None
```

Double Linked List

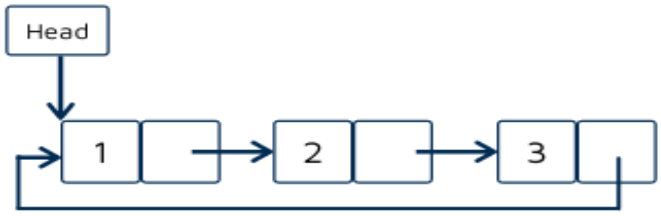
Basic Example:

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

    def __str__(self):
        return str(self.value)

    def display(self):
        print("prev:",self.prev,"value",self.value,"next:",self.next)

n1 = Node(2)
n2 = Node(4)
n3 = Node(6)
n4 = Node(8)
n1.next= n2
n2.prev = n1
n2.next= n3
n3.prev = n2
n3.next = n4
n4.prev = n3

current = n1

print('None',end='<->')
while current is not None:
    print(current.value,end='<->')
    current= current.next
print('None')
```

Advance Example:

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

class DoubleLinkList:
    def __init__(self):
        self.head = None

    def add_to_start(self,data):
        newNode = Node(data)
        # for first element
        if not self.head:
            self.head = newNode
            return

        newNode.next = self.head
        self.head.prev = newNode
        self.head = newNode

    def add_to_end(self,data):
        newNode = Node(data)
        # for first element
        if not self.head:
            self.head = newNode
            return

        last = self.head
        while last.next:
            last = last.next
        last.next = newNode
        newNode.prev=  last

    def display(self):
        current = self.head
        print('None',end=' <-> ')
        while current is not None:
            print(current.value,end=' <-> ')
            current = current.next
        print('None')
```

Circular Linked List



Circular Linked List

QUESCOL

Example:

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

class CircularLinkedList():
    def __init__(self):
        self.head = None

    def add_to_end(self,data):
        newNode = Node(data)
        if not self.head:
            self.head = newNode
            newNode.next = self.head

        current = self.head
        while current.next != self.head:
            current = current.next
        current.next = newNode
        newNode.next = self.head

    def display(self):
        if not self.head:  # If the list is empty
            print("List is empty.")
            return
        current = self.head
        while True:
            print(current.value,end=' -> ')
            current = current.next
            if current == self.head:
                break
        print()
```

```
clist = CircularLinkedList()
clist.add_to_end(4)
clist.add_to_end(5)
clist.add_to_end(5)
clist.add_to_end(5)
clist.display()

# Output: 4 -> 5 -> 5 -> 5 ->
```

Common Operations on Linked Lists

Insertion at Beginning:

Single Linked List:

```
def add_to_start(self,data):
        newNode = Node(data)
        newNode.next = self.head
        self.head = newNode
```

Double Linked List:

```
def add_to_start(self,data):
        newNode = Node(data)

        # for first element
        if not self.head:
            self.head = newNode
            return

        newNode.next = self.head
        self.head.prev = newNode

        self.head = newNode
```

Single Circular Linked List:

```
def add_to_start(self,data):
        newNode  = Node(data)

        if not self.head:
            self.head = newNode
            newNode.next  = self.head
            return

        current = self.head
        while current.next != self.head:
            current = current.next

        current.next = newNode
        newNode.next = self.head
        self.head = newNode
```

Insertion at End:

Single Linked List:

```
def add_to_end(self,data):
        new_node = Node(data)

        # for first element
        if not self.head:
            # made first element the head of linked list
            self.head = new_node
            return

        current = self.head

        # when current node has its NEXT node connected
        #  make the next node as current
        while current.next:
            current = current.next

        # making new node as the NEXT of current node
        current.next = new_node
```

Double Linked List:

```
def add_to_end(self,data):
        newNode = Node(data)

        # for first element
        if not self.head:
            self.head = newNode
            return

        current = self.head
        while current.next:
            current = current.next
        current.next = newNode
        newNode.prev=  current
```

Single Circular Linked List:

```
def add_to_end(self,data):
        newNode = Node(data)

        if not self.head:
            self.head = newNode
            newNode.next = self.head
            return

        current = self.head
        while current.next != self.head:
            current = current.next
        current.next = newNode
        newNode.next = self.head
```

Insertion After Node:

Single Linked List:

```
def add_after_node(self,prev_node_data,data):
        newNode= Node(data)

        current = self.head

        while current and current.data != prev_node_data:
            current = current.next

        if current is None:
            print('Node not found')
            return

        newNode.next = current.next
        current.next = newNode
```

Double Linked List:

```
def add_after_node(self,prev_node_data,data):
        newNode= Node(data)
        current = self.head
        while current and current.value  != prev_node_data:
            current = current.next
        if current is None:
            print('Node not found')
        newNode.next  = current.next
        newNode.prev = current
        if current.next:
            current.next.prev = newNode
        current.next = newNode
```

Single Circular Linked List:

```
def add_after_node(self,prev_node_data,data):
        newNode = Node(data)
        current = self.head
        while current and current.value != prev_node_data:
            current =  current.next
            if current ==  self.head:
                print('Node not found')
                return
        newNode.next = current.next
        current.next = newNode
```

Delete Node:

Single Linked List:

```
def delete_node(self,key):
        current = self.head

        # if head want's to be deleted
        if current and current.data == key:
            self.head = current.next
            current = None

        prev = None
        while current and current.data != key:
            prev = current
            current = current.next

        if current is None:
            print('Node not found')
            return

        prev.next = current.next
        current = None
```

Double Linked List:

```
def delete_node(self, key):
        current = self.head
        while current and current.value != key:
            current = current.next
        if current is None:
            print("Node not found")
            return
        if current.prev:
            current.prev.next = current.next
        if current.next:
            current.next.prev = current.prev
        if current == self.head:  # If it's the head node
            self.head = current.next
        current = None
```

Single Circular Linked List:

```
def delete(self,key):
        # for deleting node at start
        if self.head.value == key:
            current = self.head
            while current.next != self.head:
                current = current.next
            current.next = self.head.next
            self.head = self.head.next
        # for node other than start
        #---- we find prev and next node and for the one which wants to be  deleted
        else:
            current = self.head
            prev = None
            while  current.next != self.head:
                prev = current
                current = current.next
                if current.value == key:
                    prev.next = current.next
```

Reversing:

Single Linked List:

```
def reverse(self):
        current = self.head
        prev = None
        while current:
            nextNode = current.next
            current.next = prev
            prev = current
            current = nextNode

        self.head = prev
```

Double Linked List:

```
def reverse(self):
        current =  self.head
        while current:
            # swappiing next with prev and prev with next
            current.prev,current.next = current.next,current.prev

            # at  the end of list currents prev becomes None
            if not current.prev:
                self.head = current

            # loop backwards because of reversing next and prev
            current = current.prev
```

Single Circular Linked List:

```
def reverse(self):
        if not self.head:
            return
        current = self.head
        prev = None
        while True:
            newNode =  current.next
            current.next  = prev
            prev = current
            current = newNode

            if current == self.head:
                break
        self.head.next = prev
        self.head = prev
```

Searching:

Single Linked List:

```
def search(self,key):
        current =  self.head
        while current:
            if current.data == key:
                return True
            current = current.next
        return False
```

Double Linked List:

```
def search(self,key):
        current = self.head
        while current:
            if current.value == key:
                return True
            current = current.next
        return False
```

Single Circular Linked List:

```
def search(self,key):
        if not self.head:
            print('List is empty')
            return
        current = self.head
        while current.next  != self.head:
            if current.value == key:
                return True
            current = current.next
        return False
```

Length:

Single Linked List:

```
def length(self):
        count = 0
        current = self.head
        while current:
            count +=1
            current = current.next
        return count
```

Double Linked List:

```
def length(self):
        current = self.head
        count = 0
        while current:
            count += 1
            current = current.next
        return count
```

Single Circular Linked List:

```
def length(self):
        if not self.head:
            print('List is empty')
            return 0
        current = self.head
        count =  1
        while current.next != self.head:
            count +=1
            current=  current.next
        return count
```

Forward Traversing:

Single Linked List:

```
def display(self):
        current = self.head
        while current is not None:
            print(current.data,end=' -> ')
            current = current.next
        print('None')
```

Double Linked List:

```
def forward_trversal(self):
        current = self.head
        print('None',end=' <-> ')
        while current:
            print(current.value,end=' <-> ')
            current = current.next
        print('None')
```

Single Circular Linked List:

```
def traverse(self):
        if not self.head:  # If the list is empty
            print("List is empty.")
            return
        current = self.head
        while True:
            print(current.value,end=' -> ')
            current = current.next
            if current == self.head:
                break
        print()
```

Backward Traversing:

Single Linked List:

Not Possible

Double Linked List:

```
def backward_trversal(self):
        current = self.head
        while current.next:
            current = current.next
        print('None',end=' <-> ')
        while current:
            print(current.value,end=' <-> ')
            current = current.prev
        print('None')
```

Single Circular Linked List:

Not Possible