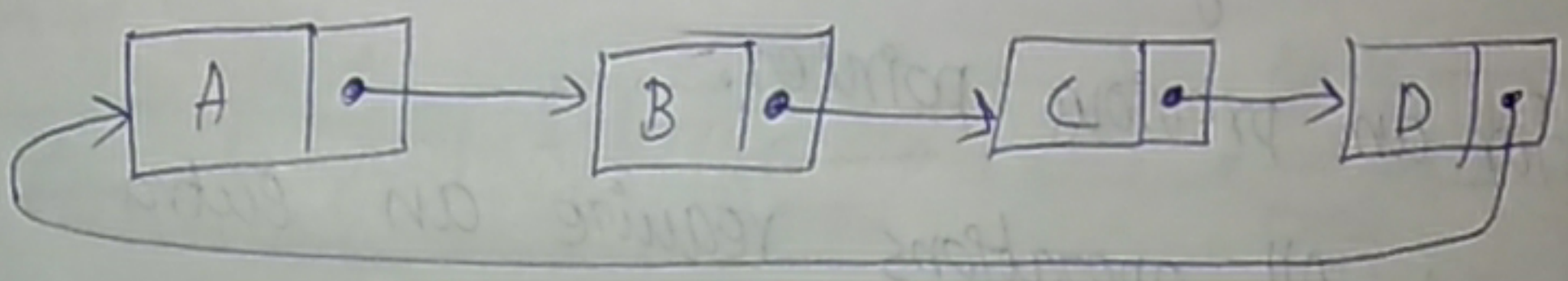
Day 11

Circular linked list

* Circular linked list is a linked list where all are connected to from a circle. There is no null at the end.

* A circular linked list where all nodes can be a singly linked list or doubly linked list.



Advandage :-

* Any node can be a starting point

* Circular lists are mainly useful in applications to repeadly go around the list.

Singly list implementation

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

class linked list :
    def __init__(self, head):
        self.head = None

    def push(self, new_data):
        new_node = Node(new_data)
        new_node.next = self.head
        self.head = new_node

    def insertafter(self, prev_node, new_data):
        if prev_node is None:
            print("the given node must in
                                linked list")
            return
```

```
        new_node = Node(new_data)
        new_node.next = prev_node.next
        prev_node.next = new_node

def append(self, new_data)
        new_node = Node(new_data)
        if self.head is None:
            self.head = new_node
            return
        last = self.head
        while (last.head next):
            last = last.next
        last.next = new_node

def printlist(self):
        temp = self.head
        while (temp):
            print(temp.data), €
            temp = temp.next
```

```
L = linked list()
L.append(6)
L.push(7)
L.push(1)
L.append(4)
L.InsertAfter(L.head.next, 8)
L.print list()
```

Output :

1 7 8 6 4

Insertion in Doubly linked list :

```
class Node:

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

```
class doublylinked list:
    def __init__(self):
        self.head = None
    def push(self, new_data):
        new_node = Node(new_data)
        new_node.next = self.head
        if self.head is not None:
            self.head.prev = newnode
        self.head = new_node
def insertafter(self, prev_node, new_data):
    if prev_node is None:
        print("the given previous node
              cannot be Null")
        return
    new_node = Node(new_data)
    new_node.next = prev_node.next
    prev_node.next = new_node
    new_node.prev = prev_node
If new_node.next is not None:
        new_node.next.prev = new_node
```

```
def append(self, new_data):
    new_node = Node(new_data)
    new_node.next = None
    If self.head is None:
        new_node.prev = None
        self.head = new_node
        return
    last = self.head
    while(last.next is not None):
        last = last.next
    last.next = new_node
    new_node.prev = last
    return

def printlist(self, node):
    print("\n traversal in forward direction")
    while (node is not None):
        print("%d" %(node.data)),
        last = node
        node = node.next
print("\n traversal in reverse direction")
while (last is not None):
    print("%d" %(last.data),
        last = last.prev
```

```
L = doubly linkedlist()
L.append(6)
L.push(7)
L.push(1)
L.append(4)
L.Insertafter(L.head.next, 8)
L.printlist(L.head)
```

Output:

Traversal in forward direction:

1 7 8 6 4

Traversal in Reverse direction:

4 6 8 7 1