Lecture : 1

* Data Structure : a way data is stored in a computer.

→ Types : Arrays (list, dictionary, tuple),
Linked Lists, Queues, Stacks

→ Array : collection of a fix number of items of same data type. (element + index)

→ Linear Array : always the starting value would take place at index 0.

List [l=[ ] → l.append[10] / on l=[10, 20, 30]

Array [define length first; a = [None] * 10]

for in range (len(a)):
a[i] = value

print → for i in range(len(a)):
print(a[i])

① Task (PRINT)

do this using while loop for reverseprint
i = len(a)-1
while i>=0:
print(a[i])
i -= 1

```
def printArray(b):
    for i in b:
        print(i)

a = [10, 20, 30]
printArray(a)        ↳ passing reference
print(a[0])  → 100 → not 10
b[0] = 100
```

```
class A():
    def ——
        ——

a = [10, 20, 30]
t1 = A()
t1.printArray(a)
```

```
class A():
    def __init__(self, a):
        self.a = a

    @classmethod
    def printA(cls, a):
        ——
        ——

a = [10, 20, 30]
t1 = A(a)
A.printArray()
```

* No Built-in Function (except for len())

② Task (INSERTION)

extra taken since dynamically length cannot be changed.

10	20	30	40	50	0	0	0
0	1	2	3	4	5	6	7

insert 200 in index 2

10	20	200	30	40	50	0	0
0	1	2	3	4	5	6	7

def insert(arr, index, value, size) → valid value in arr (here size = 5)

to check if empty space

```
if (size == len(arr)):
    print("no space in array")
    return
```

check valid index

```
if (index < 0 or index > size):
    print("wrong index")
    return
```

index can not be more than 5 as আমরা empty space রাখবো না.

Beginner level Array Task:

① Reverse the array (outplace)

10	20	30	40	50
0	1	2	3	4

i

→

50	40	30	20	10
0	1	2	3	4

j

```
def reverse(a):
    newA = [0] * len(a)
    ~~for i in range(len(a)):~~ can't as not more than 1 condition
    ~~while~~ i = 0 ; j = ~~newA.length-1~~ len(newA) - 1
    while ~~i<a.length~~ i < len(a) :
        new[j] = a[i]
        i += 1
        j -= 1
    return newA
```

```
a = ____
~~b~~ = reverse(a)
print for ~~b~~ → ~~a gets change~~
```

② Reverse Array (In place)

```
def reverse(a):
    ~~for i in range(len(a)):~~ i = 0 ; j = len(a) - 1
    while ~~i < len(a)//2~~ i < j :
        temp = a[i]
        a[i] = a[j]
        a[j] = temp
        i += 1
        j -= 1
```

```
a = ____
reverse(a)
```

③ Copy Array
④ Resize Array
] Home Work using Outplace

Linear Vs Circular

10	20	30	40	50	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9	10

} Linear

circular size=5

0	1	2	3	4	5	6	7	8	9	10	
0	0	10	20	30	40	50	0	0	0	0	→ starts at 2
0	0	0	0	10	20	30	40	50	0	0	→ starts at 4
50	0	0	0	0	0	0	10	20	30	40	→ at 7
40	50	0	0	0	0	0	0	10	20	30	→ at 8

Properties of Circular Array

1. start index
2. size (number of valid elements)

* C.A is not a built-in structure. We need to implement it from a linear Array.

1 Forward Printing (Linear)

```
for i in range(size):
    print(lin[i])
```

→ এখানে 'i' both loop controller and index

Forward Printing (Circular) (start=4)

```
index = start
for i in size:
    print(c[index])
    index = index + 1
```

i	index	output
0	4	10
1	5	20
2	6	30
3	7	40
4	8	50
5	9	X

→ size এর সমান

```
index = start
for i in size:
    print(c[index])
    index += 1
    if (index == len(c)):
        index = 0
```

shortcut

Hint: len(a) এর মান 11, 11%11=0

→ `index = (index+1) % len(c)`

start = 8 →

i	index	output
0	8	10
1	9	20
2	10	30
3	11	error

loop চলবে ৪ বার but still error

(index out of bound error)

→ i[11]

start = 8

i	index	output
0	8%11=8	10
1	9%11=9	20
2	10%11=10	30
3	11%11=0	40
4	1%11=1	50
5	2%	

② Reverse Printing (size=5)

10	20	30	40	50	0	0	0	0	0	0	→ stant = 0
0	1	2	3	4	5	6	7	8	9	10	

last index = 4 = size - 1

0	1	2	3	4	5	6	7	8	9	10	
0	10	20	30	40	50	0	0	0	0	0	→ stant = 1 last = 5 = (size-1) + 1
0	0	10	20	30	40	50	0	0	0	0	→ st = 2 last = 6 = (size-1) + 2
0	0	0	10	20	30	40	50	60	0	0	→ st = 3 last = 7 = (size-1) + 3

∴ last index = (size - 1) + stant = stant + size - 1

40	50	0	0	0	0	0	0	10	20	30	st = 8 sz = 5 len = 11
0	1	2	3	4	5	6	7	8	9	10	

last = st + sz - 1 = 8 + 5 - 1 = 12 → out of bound

∴ last index = (stant + size - 1) % len(c)

= 12 % 11 = 1

↳ thumb rule 2

index = (stant + size - 1) % len(c)

```
for i in size:
    print(c[i])
    index = index - 1
    if (index < 0):
        index = len(c) - 1
```

(3) Convert linear to circular:

```
def circularize (lin, st, sz):
    cin = [0] * len(lin)
    index = st
    for i in sz:
        cin[index] = lin[i]
        index = (index+1) % len(cin)
    return cin
lin = [10, 20, 30, 40, 50, 0,0,0,0,0]
cin = circularize (lin, 8, 5)
print(cin)
```

(4) Resize

```
def resize(cin, st, sz):
    resCin = [0] * (len(cin)+2)  → on can take parameter.
    indexcin = stant
    index_resCin = stant
    for i in sz:
        resCin [index_resCin] = cin[indexcin]
        indexcin = (indexcin +1) % len(cin)
        index_resCin = (indexresCin +1) % len (resCin)
    return resCin
```

0	1	2	3	4	5	6	7
0	80	20	30	40	0	0	0

→ st = 2
sz = 3

0	1	2	3	4	5	6	7	8	9
0	0	20	30	40	0	0	0	0	0

st = 2
sz = 3

Thumb Rules.

(1) C.A -Inaverse করতে, one var. নিবে for loop controller আরেকটি diff var. for circular annayn index.

(2) Whenever you ~~create~~ increment the index of a c.a, always mod the incremented value with c.a's length.

(3) When you decrement index of c.a. always check whether it becomes negative. হলে it point to end of the array.

Multidimensional Array:

an array with more than two or equal dimensions. (rows and column)

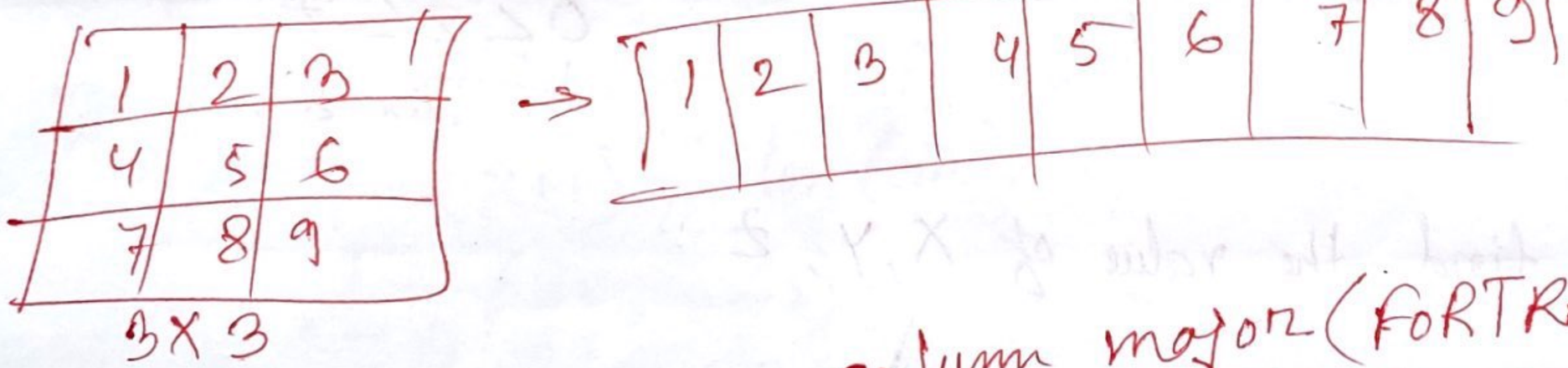
→ it gives us natural way to deal with real-world objects that are multidimensional.

→ You can use it it image rendering.

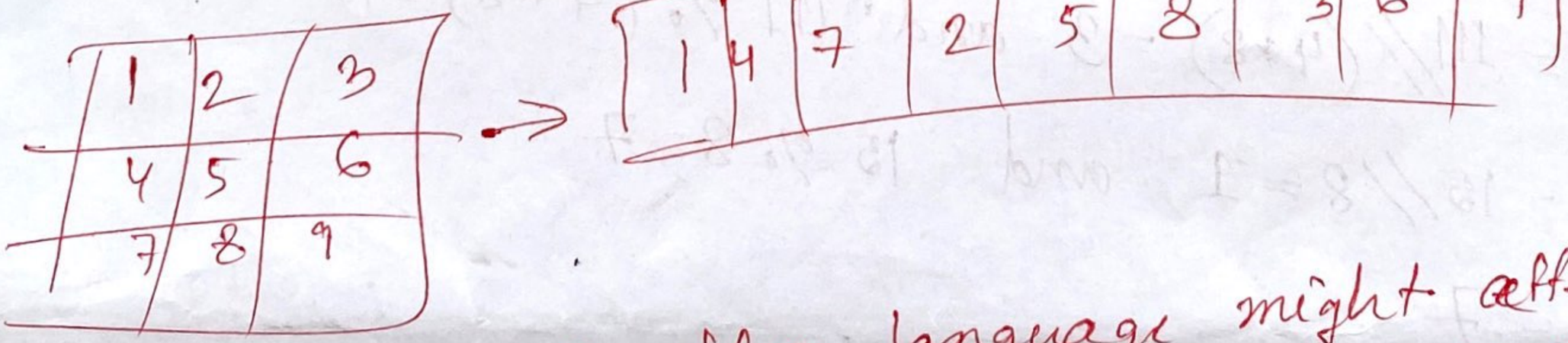
* Programing languages store M.A as linear arrays in RAM.

→ it first place all elements of a single dimension to the ~~array~~ index by increasing ~~into~~ values ~~then~~ in consequtive locations in memory. then it traverse other direction. and does the same thing.

row major (Java, C) → More Common



column major (FORTRAN)



→ using one way to other language might affect the performance but it will work better for that language.

→ How to determine the translated linear index of a M. D index from original array?

4D → M×N×O×P (dimensions) of box array

element at the index you are interested in box[w][x][y][z]

Then the index of that element in linear array:

$$W \times (N \times O \times P) + x \times (O \times P) + y \times P + z$$

suppose, linear array of length 128 store 3D array of dimensions 4×4×8. What are the M.D indexes of element stored at 111?

so, 111 इस linear का index. Need to find M.D का index.

$X * (4 * 8) + Y * (8) + Z = 111$; here; $\underbrace{0 <= X <= 3}_{\text{since dim. 4}}$

$0 \leq Y \leq 3$

$\underbrace{0 \leq Z \leq 7}_{\text{dim 8.}}$

Now to find the value of X, Y, Z:

first,

$X = 111 // (4*8) = 3$ and $111 \% (4*8) = 15$

$Y = 15 // 8 = 1$ and $15 \% 8 = 7$.

$Z = 7$

so, the dimension of linear index 111 is [3][1][7]

$X * (4*8) + Y * (8) + Z = 96$

$X = 96 // (4*8) = 3$ and $96 \% (4*8) = 0$

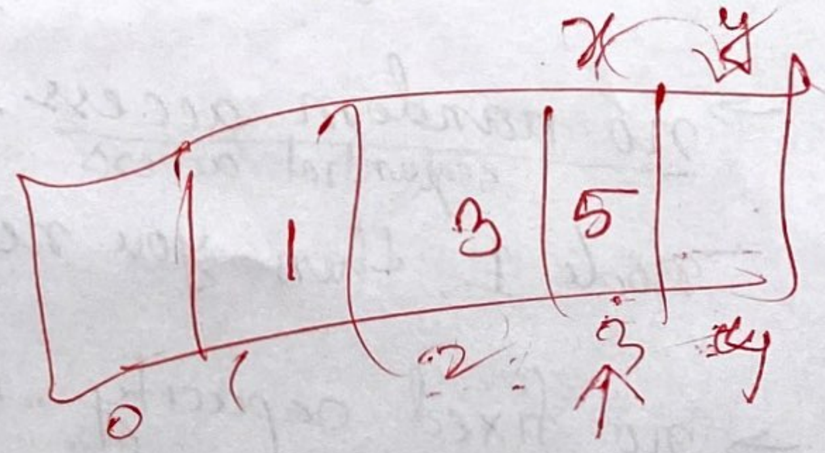
$Y = 0 // (8) = 0$ and $0 \% 8 = 0$

$Z = 0$

$96 = [3][0][0]$

$a = [[0]*2, [0]*2]$

```
def insert(cin, item, st, sz, ind):
    if (sz < len(cin)):
        x = (start + size - 1) % len(cin)
        y = (x+1) % len(cin)
        while (y != index):
            cin[y] = cin[x]
            x = x-1;
            if (x<0):
                x = 0
            y = y-1;
            if(y<0):
                y = 0
        if y = ind
    else:
        resize()
        repeat
```



1 (2)

cin[y] = item

Linked List

* Arrays:

adv: ① Random Access (I can access the array's element in constant time → a[2])

② Less memory space

disadv: ① fixed capacity (declare array with fixed length, after/beyond that capacity nothing can be added; for that we need to create new array) and contiguous memory allocation.

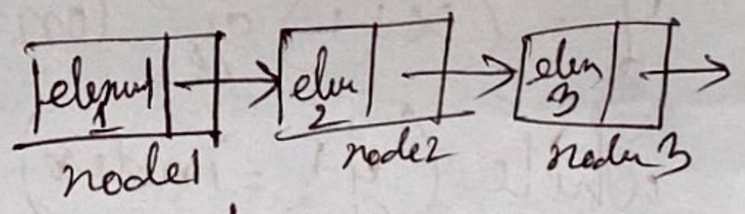
② Shifting (requires much more operations while we do insertion or removal)

* These limitations can be overcome with linked list.

Linked list:

→ a datastructures which is created with a set of nodes and these nodes are connected with links.

→ nodes: an object that has a data and the reference/address of the next node.



→ no random access. If you want to access node 3 from node 1 then you need to traverse through node 2.
(sequential access)

→ no fixed capacity. Add as many as you need, just you need to connect them with the last node.

→ required more memory space (one for value and another for link (address of next node))

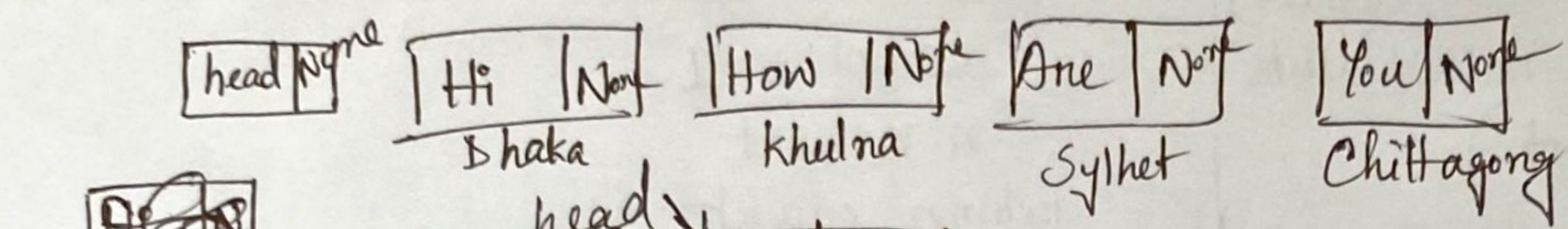
→ starting/first node is called head reference

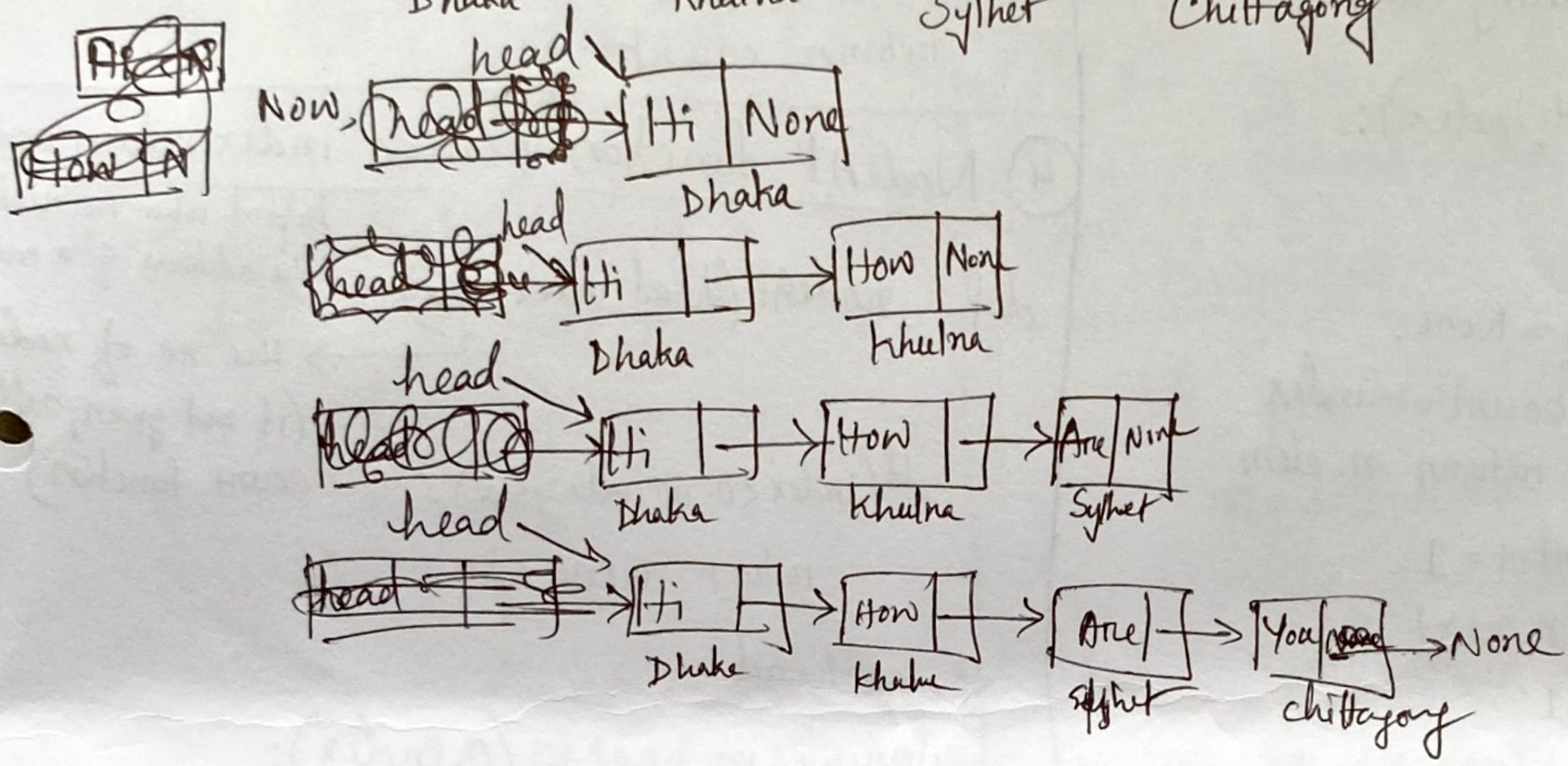
→ linked list doesn't follow contiguous memory allocation. (Dynamic allocation as nodes are object). The nodes can be anywhere in the linked list and the connections are made through links/next node reference.

Node class

```
class Node:
    def __init__(self, elem, next):
        self.elem = elem
        self.next = next
```

elem → data
next → next node's reference





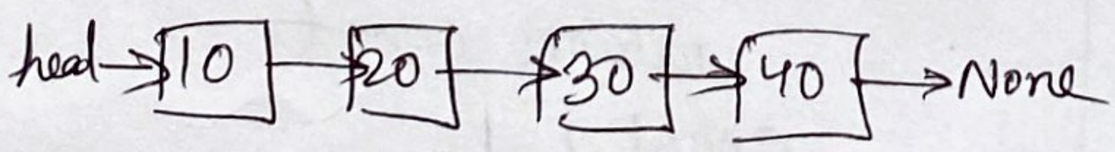
Tester

```
head = None
#creating the nodes
n1 = Node('10', None)
n2 = Node('20', None)
n1.next = n2
#Assigning the head reference to list
head = n1
```

another way to create by directly calling constructure (reverse order)

```
head = None
n4 = Node('40', None)
n3 = Node('30', n4)
n2 = Node('20', n3)
n1 = Node('10', n2)
head = n1
```

① Iterating through list:



starting point → head
ending point → none

we don't have length of linked list here; so how do you know where to stop? see the last node's next reference is none.

in every iteration we have to go to next node from current one with next reference.

```
n=head
while n!=None:
    #do something
    n=n.next
```

② Count Nodes

*how many nodes in list?

→ taking head reference will automatically give me the access to list. (by next reference)

```
def countNode(head):
    count=0
    n=head
    while n!=None:
        count+=1
        n=n.next
    return count
```

③ Get Function (Given index, return element)

but there isn't any index in linkedlist, how to get it? (using count)

```
def get(head,index):
    count=0
    n=head
    while n!=None:
        if count==index
            return n.elem
        count+=1
        n=n.next
    return -1
```

tester

get(head,2) → output = 30

get(head,10) → output = -1 (invalid index

④ NodeAt function (Given index, return node)

helpful when we need the address of a node.

```
def nodeAt(head,index,size):
    if(index<0 or indx≥size):
        return None
    n=head
    for i in range(0,index):
        n=n.next
    return n
```

→ the no. of nodes. (if not given, call count function)

tester

nodeAt(head,2,4) → output 30

nodeAt(head,10,4) → output None

⑤ Set function (Given index, element set it to the linkedlist)

```
def set(head,index,elem):
    count=0
    n=head
    while n!=None:
        if count==index:
            n.element=elem
        count+=1
        n=n.next
```

tester

set(head,2,50)

output → 10 20 50 40

⑥ Search (takes elem and return index)

```
def indexof(head,elem):
    count=0
    n=head
    while n!=None:
        if n.element==elem:
            return count
        count+=1
        n=n.next
    return -1
```

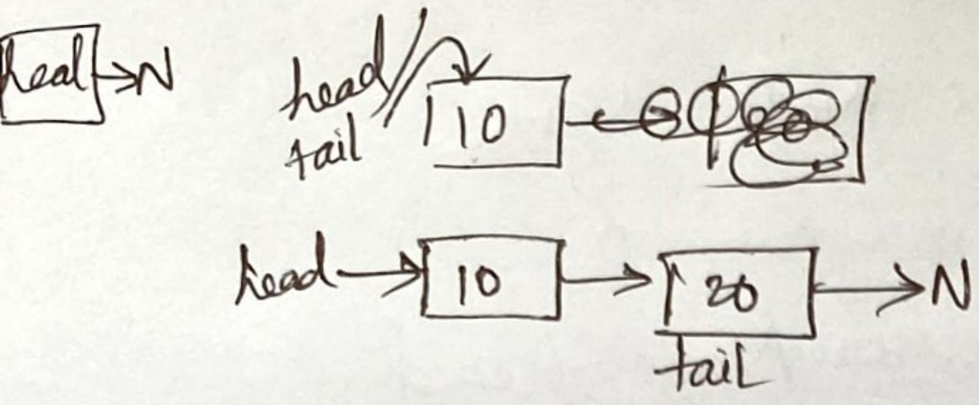
tester

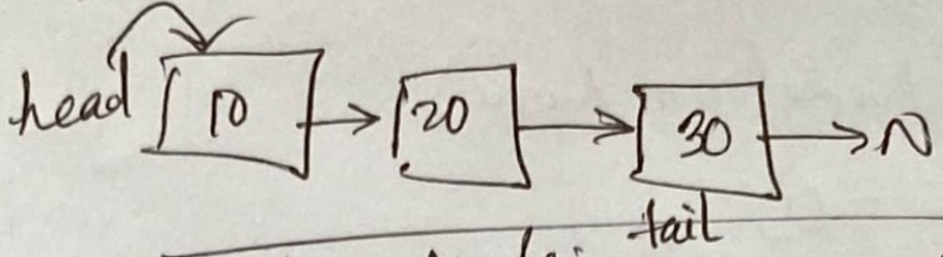
indexOf(head,50) → output = -1

indexof(head,40) → output = 3

(7) Creating Linked List from array

* take two variables (head, tail)
* at first head and tail will refer to the same node (1st node). then keep head fixed at first node and change tail (after connecting another node that would be new tail.)






```
class LinkedList:
    def __init__(self, a):
        self.head = None
        tail = None
        for i in a:
            n = Node(i, None):
            if (self.head == None):
                self.head = n
                tail = n
            else:
                tail.next = n
                tail = n
```

```
class Node:
    def __init__(self, elem, next):
        self.elem = elem
        self.next = next
```

(8) PrintList

```
def printLL(self):
    n = self.head
    while n != None:
        print(self.elem)
        n = n.next
```

(9) CopyList

```
def copyList(head):
    copyhead = None
    copyTail = None
    n = head
    while n != None:
        newnode = Node(n.element, None)
        if (copyHead == None):
            copyHead = newNode
            copyTail = newNode
        else:
            copyTail.next = newNode
            copyTail = newNode
        n = n.next
```

test

```
l1 = [1, 2, 3, 4, 5]
l2 = LinkedList(l1)
l2.printLL()
```

(10) Insert at given index

```
def insert(head, size, elem, index):
    if(index<0 or index>size):
        print("Invalid")
    newnode = Node(elem, None)
    if (index==0):
        newNode.next=head
        head=newNode
    else:
        pred=NodeAt(index-1)
        newNode.next= pred.next
        pred.next = newNod
    return head.
```

(11) Remove (given index, remove node)

```
def remove(head, size, index):
    if (index<0 or index>=size)
        print("invalid")
    removeNode = None
    if (index==0):
        removeNode =head
        head= head.next
    else:
        pred =nodeAt(index-1)
        removeNode = pred.next
        pred.next= removeNode.next
    removeNode.elem = None
    removeNod.next = None
    return head.
```

(12) ReverseList (out-place)

```
def reverseO(head):
    copyHead=None
    n= head
    while (n!=None):
        newNode = Node(n.elem, None)
        if(copyHead==None):
            copyHead=newNode
        else:
            newNode.next=copyHead
            copyHead=newNode
        n=n.next
    return copyHead
```

(13) Reverse (in place)

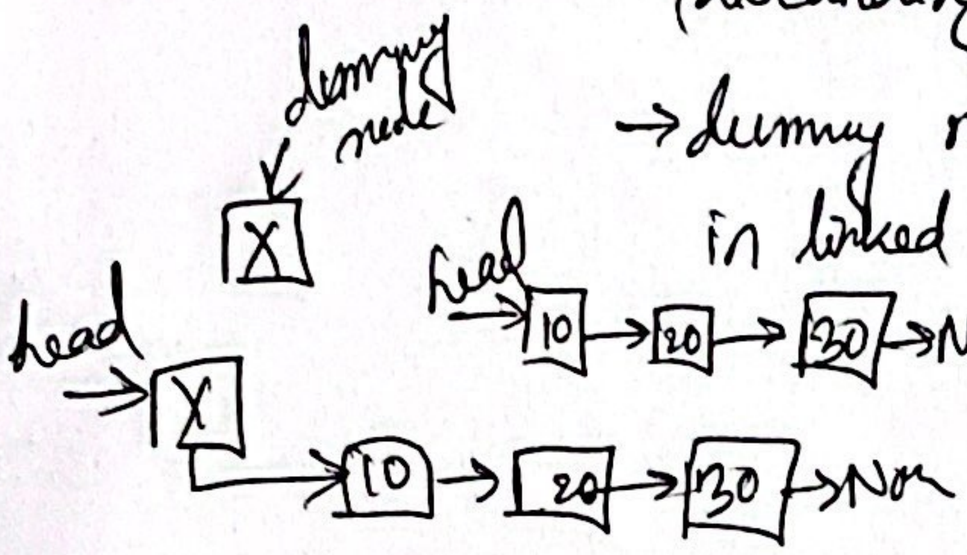
```
def reversI(head):
    newHead=None
    n= head
    while(n!=Now):
        nextNode=n.next
        n.next=newHead
        newHead = n
        n = nextNode
    return newHead
```

(14)

```
def rotateLeft(head):
    oldHead=head
    head=head.next
    tail=head
    while(tail.next!=None):
        tail=tail.next
    tail.next=oldHead
    oldHead.next=None
    return
```

Types of Linked List

⇒ Dummy Nodes: a node with any value (element) in it.
→ mainly used to omit the head changing step.
(discarding the special case of insertion and removal)
→ dummy nodes is something that will replace over head in linked list.



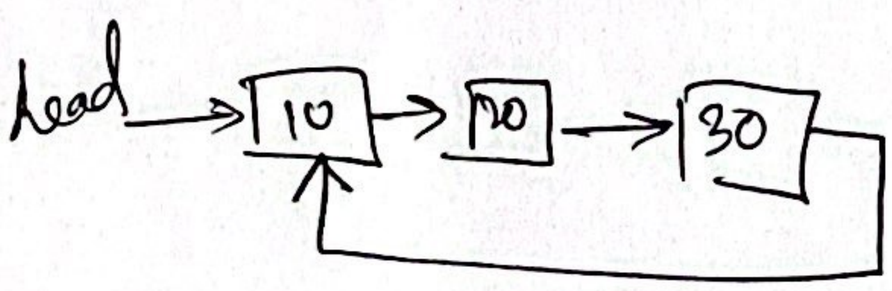
(I) Singly Linked List

```
n = head
while n != None:
    #do something
    n = n.next
```

(II) Dummy head singly linked list

```
n = head.next
while n != None:
    #do something
    n = n.next
```

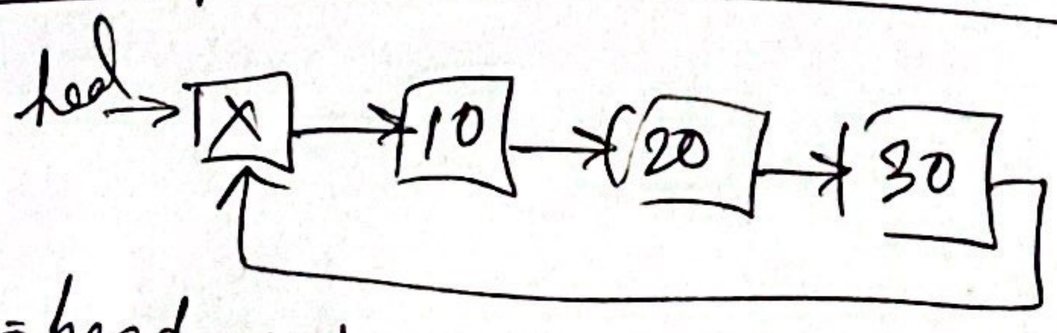
(III) Singly Circular linked list



last node points to the first one.

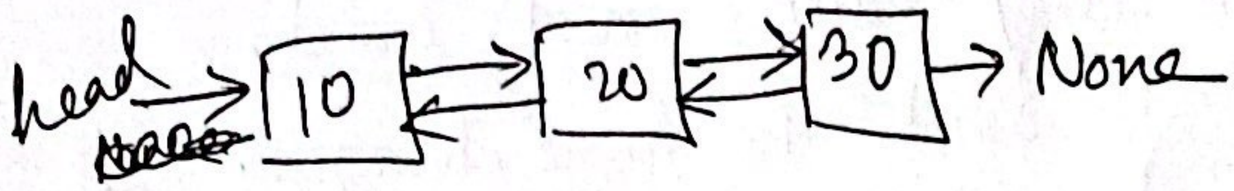
```
n = head
while n.next != head:
    #do something
    n = n.next
```

(IV) Dummy headed Circular Singly linked list.




```
n = head.next
while n.next != head:
    #do something
    n = n.next
```

(V) Doubly Linked List:



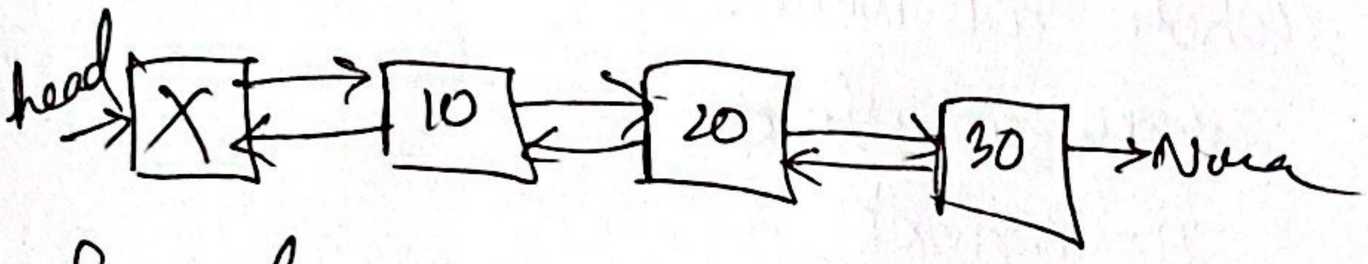
Forward iteration

```
n = head
while n != None:
    # do something
    n = n.next
```

Backward iteration

```
n = tail
while n != None:
    # do something
    n = n.prev
```

(VI) Dummy headed Doubly Linked List.



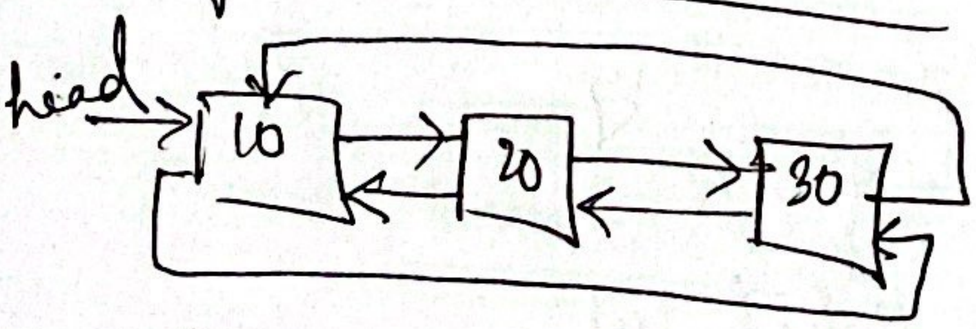
forward

```
n = head.next
while n != None:
    # do something
    n = n.next
```

Backward

```
n = tail
while n.prev != None:
    # do something
    n = n.prev
```

(VII) Doubly Circular Linked List



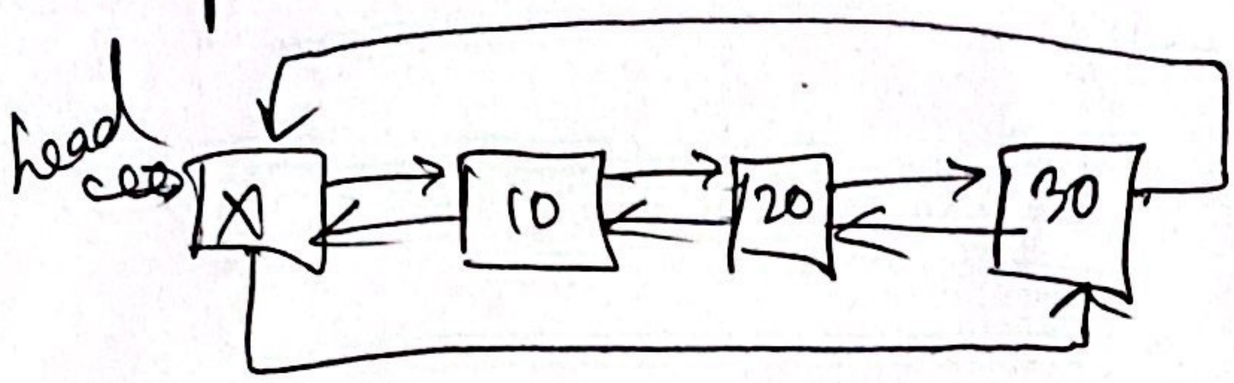
forward

```
n = head
while n.next != head:
    # do something
    n = n.next
```

backward

```
n = head.prev
while n.prev != head:
    # do something
    n = n.prev
```

(VIII) Dummy headed Doubly Circular Linked List



forward

```
n = head.next
while n != head
    # do something
    n = n.next
```

backward

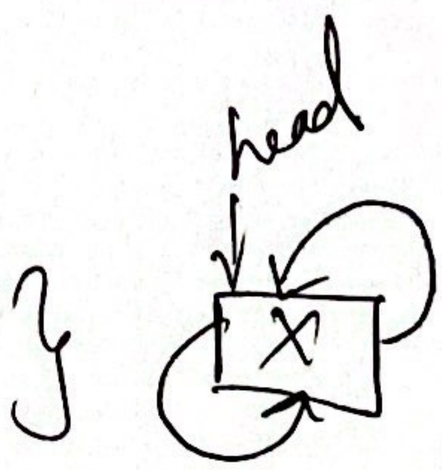
```
n = head.prev
while n != head:
    # do something
    n = n.prev
```

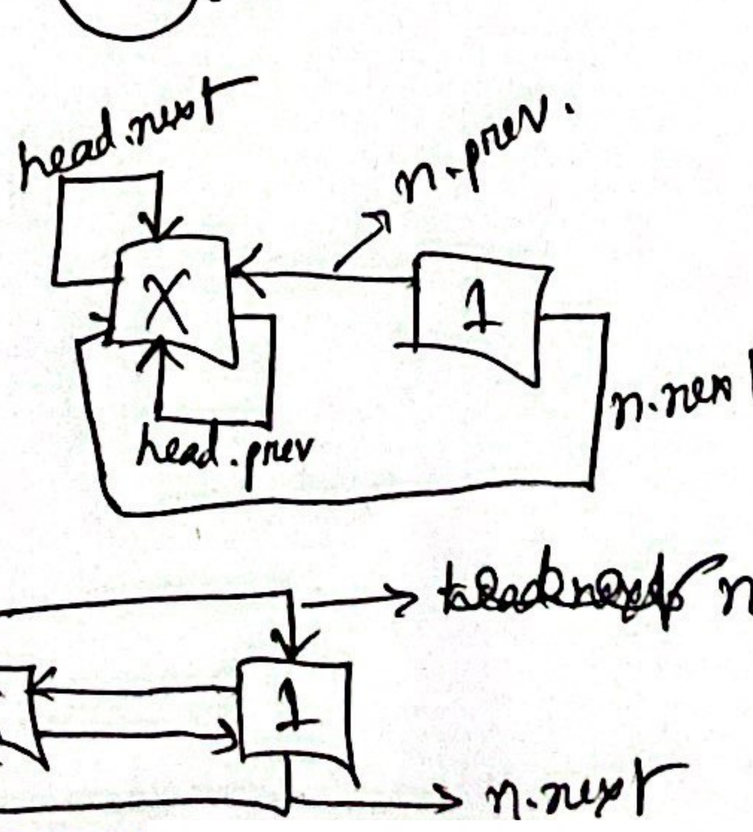
for DHDCLL

```
Class Node:
    def __init__(self, elem, next, prev):
        self.elem = elem
        self.next = next
        self.prev = prev.

Class DHDCLL:
    def __init__(self):

# create dumy node
    head = Node(None, None, None)
# make it circular
    head.next = head.prev = head
# adding first nod.
    n = Node(1, None, None)
    n.next = head.next
    n.prev = head
    head.next = n
    n.next.prev = n
```





(i) Insert (in DHDCLL) : Given a node and elem after that

insert

```
def insertAfter(self, p, elem):
    n = Node(elem, N, N).
    q = p.next
    n.next = q
    n.prev = p
    p.next = n
    q.prev = n
    return n
```

(ii) Remove (Given node, remove it)

```
def removeNode(self, n):
    p = n.prev
    q = n.next
    p.next = q
    q.prev = p
    n.next = n.prev = None
    n.element = None
```