Date:

Grant Thornton
An instinct for growth™

Linear array

len(arr) = 6
k = 3 = 3

Shift left

k = 3

10	20	30	40	50	60
40	20	30	0	50	60
40	50	30	0	0	60
40	50	60	0	0	0

```
def shift_left(source, k):
    for i in range(len(source)-k):
        source[i] = source[i+k]
        source[i+k] = 0
    return source.
```

Rotate Left

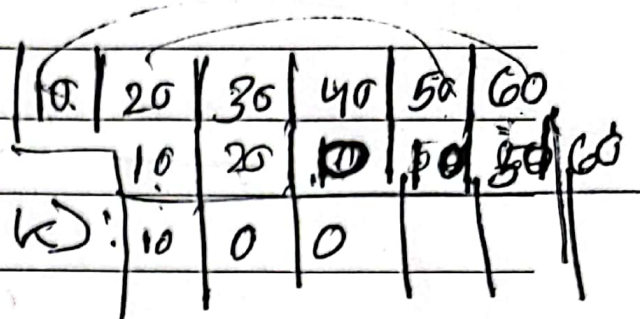
10	20	30	40	50	60
40	20	30	10	50	60
50	50	30	10	20	60
60	50	60	10	20	30

```
def rotate_left(source, k)
    for i in range(len(source)-k):
        temp = source[i]
        source[i] = source[i+k]
        source[i+k] = temp
    return source
```

Date:

Grant Thornton
An instinct for growth™

Shift Right




```
def shift_right(source, k):
    i = len(source) - 1
    while i >= k:
        source[i] = source[i-k]
        source[i-k] = 0
    return source
```

i = 1

Rotate Right

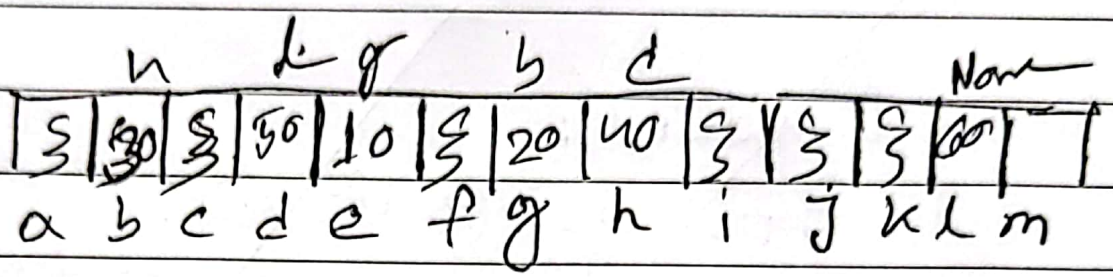
```
def rotate_right(source, k):
    i = len(source) - 1
    while i >= k:
        temp = source[i]
        source[i] = source[i-k]
        source[i-k] = temp
        i -= 1
    return source
```

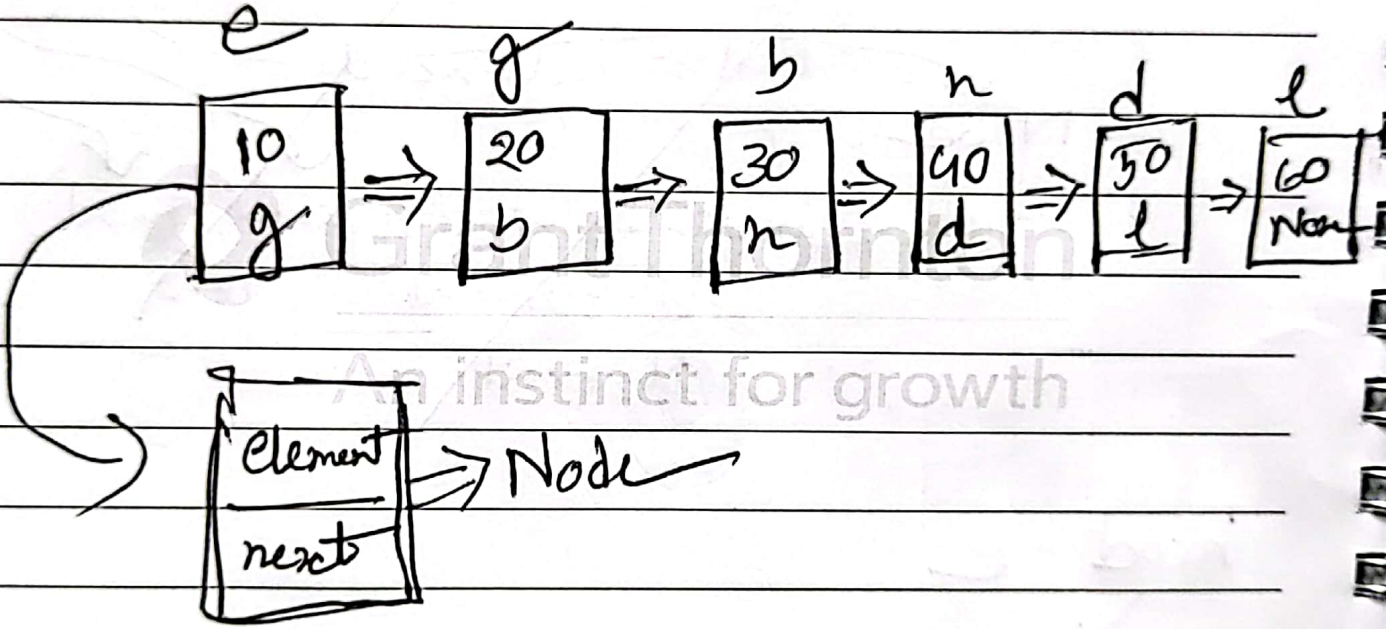
Date:

Grant Thornton
An instinct for growth™

Linked list

10, 20, 30, 50, 60





Date:

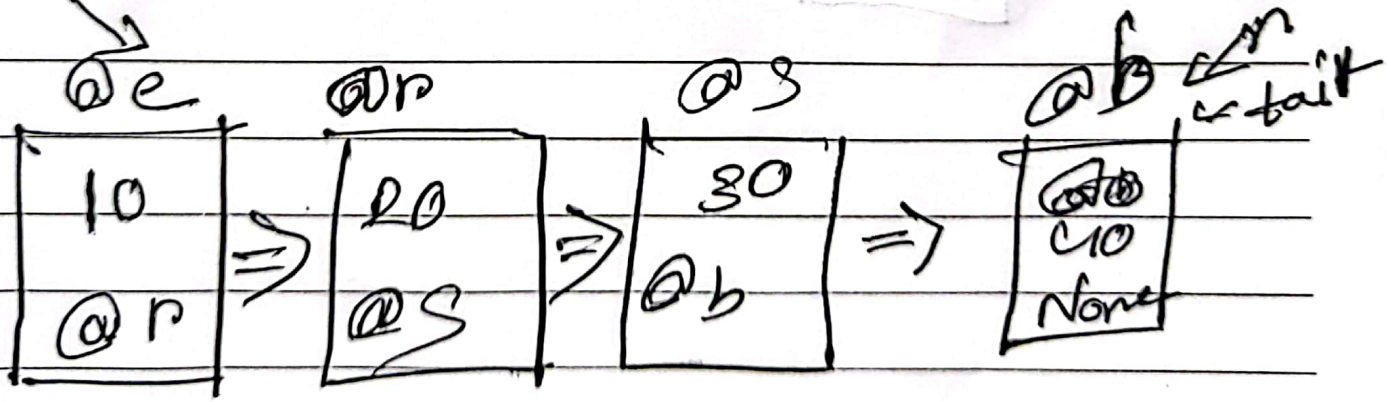
Grant Thornton
An instinct for growth™

a = [10, 20, 30, 40]

```
class LinkedList:
    def __init__(self, a):
        self.head = Node(10, None)
        tail = self.head

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

head = e



Construction of a Linked List

Grant Thornton
An instinct for growth™

a = [10, 20, 30, 40]

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

class LinkedList:
    def __init__(self, a):
        self.head = Node(10, None)
        tail = self.head
        i = 1
        while i < len(a):
            n = Node(a[i], None)
            tail.next = n
            tail = n
            i = i + 1

    # Print
    def printList(self):
        n = self.head
        while (n != None):
            Print(n.element)
            n = n.next
```

Traversal of a linked list

Date:

Count

```
def countNode(self):
    n = self.head
    count = 0

    while n != None:
        count += 1
        n = n.next
    return count
```

Date:

Node at

Validation check is must

```
def nodeAt (self, idx)                    # count function
    if idx<0 or idx>= self.countNode():
        return
    n = self.head
    count = 0
    while n != None:
        if count == idx:
            return n
        count += 1
        n = n.next
```

Pred = Predecessor

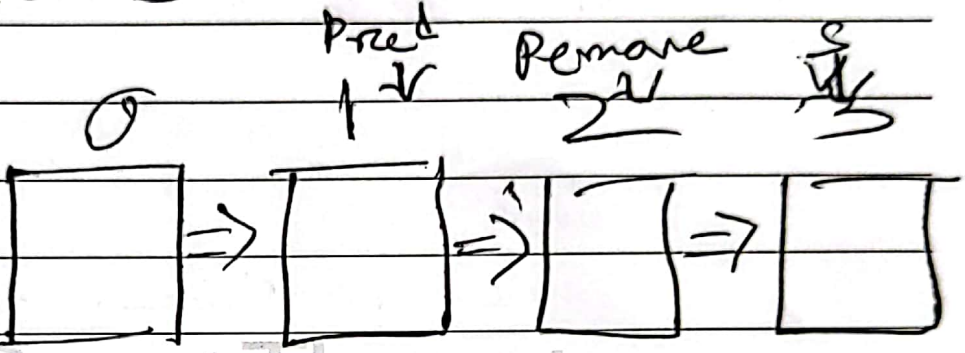
S = Successor

Remove

```
def remove (self, idx):
    if idx<0 or idx>= self.countNode():
        return
```

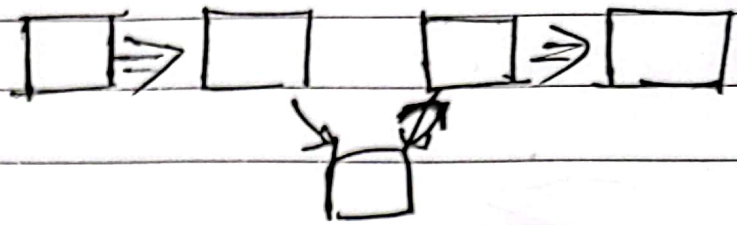
Date:


```
if idx == 0 :
    rem = self.head
    self.head = self.head.next
else:
    Pred = self.nodeAt(idx-1)
    rem = pred.next
    s = rem.next
    pred.next = s
```



1st find pred,
set variable for remove
then set successor
then connect pred to s

Date:



insert

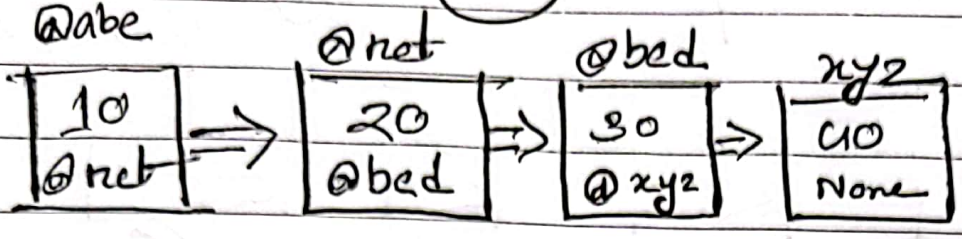
```
def insert(self, idx, elem):
    newnode = Node(elem, None)
    if idx < 0 or idx >= self.countNode():
        return

    if idx == 0:
        newnode = self.head
        self.head = self.head.next
    else:
        pred = self.nodeAt(idx-1)
        S = pred.next
        pred.next = newnode
        newnode.next = S
```

Date:

Grant Thornton
An instinct for growth

Doubly Linked List



non dummy headed Singly linear Linked List

Class Node:

Constructor

```
def __init__(self, elem, p, n):
    self.element = elem
    self.next = n
    self.prev = p
```

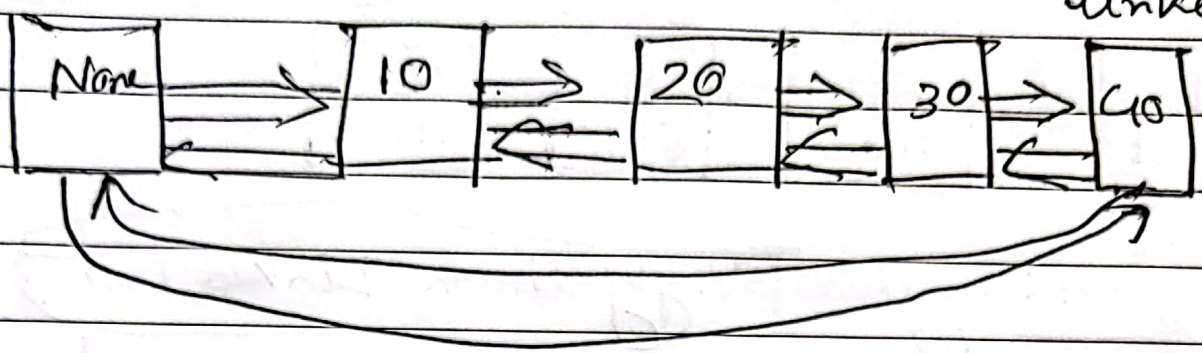
Class DoublyList:

```
def __init__(self, a):
```

```
def printList(self):
    print(self.head.element)
    n = self.head.next
    while n != self.head:
        print(n.element)
        n = n.next
```

Date:

Dummy headed doubly circular linked list




```
def printList(self):
    n = self.head.next
    while n != self.head
        print(n.elem)
        n = n.next
```



```
def printRev(self)
    n = self.head.prev
    while n != self.head
        print(n.elem)
        n = n.prev
```

Date:

Grant Thornton
An instinct for growth™

```
def countNode(self):
    count = 0
    n = self.head.next
    while n != self.head:
        count += 1
        n = n.next
    return count
```

```
def nodeAt(self, idx):
    if idx < 0 or idx >= self.countNode():
        return
```

```
def nodeAt(self, idx):
    if idx < 0 or idx >= self.countNode():
        return
    count = 0
    n = self.head.next
    while n != self.head:
        if count == idx:
            return n
        count += 1
        n = n.next
```

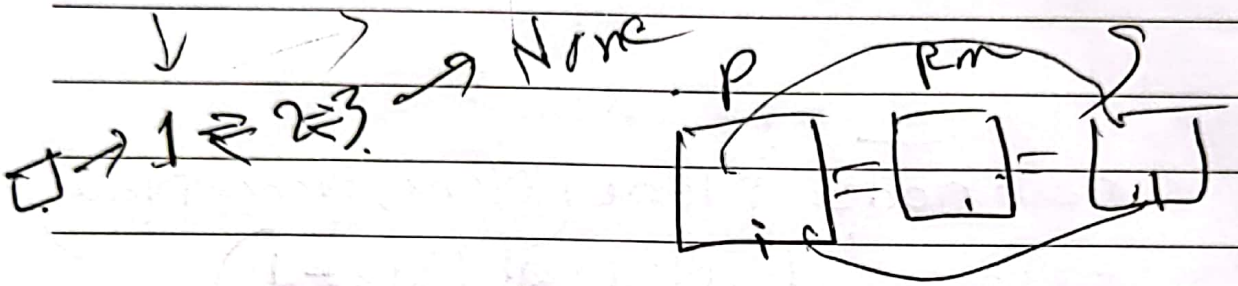
Constructor

```
Class DHDLL:
    def __init__(self, a):
        self.head = Node(None, None, None)
        tail = self.head
        i = 0
        while (i < len(a)):
            n = Node(a[i], none, none)
            tail.next = n
            n.prev = tail
            tail = n
            i += 1

        tail.next = self.head
        self.head.next = tail
```

Date:


```
def remove (self, idx):
    if idx < 0 or idx >= self.countNode():
        return
```




```
    rem = self.nodeAt(idx)
    pred = rem.prev
    S = rem.next
    pred.next = S
    S.prev = pred
    rem.next = rem.prev = rem.elem = None
```

None → 1 ⇄ 2 ⇄ 3 → None

```
    n = head
    while (n.next != None):
        if n.next.element % 2 == 0:
            n.next = n.next.next
            n.next.prev = n
        else:
            n = n.next
```

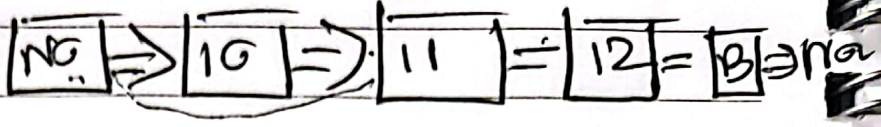
Date:

Grant Thornton
An instinct for growth™

```
def insert (self, idx, item):
    if idx < 0 or idx >= self.countNode():
        return

    newnode = Node(None, None, None)
    pred = self.NodeAt(idx-1)
    ~~newnode = pred.next~~
    S = pred.next
    newnode.next = S
    S.prev = newnode
    newnode.prev = pred
    pred.next = newnode
```

Date:



```
def remove(self, idx):
    if idx<0 or idx>=self.count():
        return
    n = head
    while n.next != None:
        if n.next.elem %2==0:
            n.next = n.next.next
            n.next.prev = n
        else
            n = n.next
            pred = n.prev
    rem = n.next
    pred = rem.prev
    s = rem.next
    pred.next = s
    s.prev = pred
```

Date:


```
def remove(self, idx):
    if idx<0 or idx>=0:
        return
    n = head

    while n.next != None:
        if n.next.elem % 2 == 0:
            rem = n.next
            pred = rem.prev
            S = rem.next
            pred.next = S
            S.prev = pred

            rem.next = rem.prev = rem.elem
                                = None
        else:
            n = n.next

    return head
```

Date:

Grant Thornton
An instinct for growth

Circular Array

a = [10, 20, 30, 40, 50]

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

Start = 4
Size = 5

```
class CircularArray:

    def __init__(self, a, start, size):
```

Date:


```
def printForward(self):
    i = 0
    index = self.start
    while i < self.start:
        print(self.c[index])
        i += 1
        index = (index+1) % len(self.c)
```

```
def printReverse(self):
    i = 0
    lastidx = (self.start+self.size-1) % len(self.c)
    while i < self.size:
        print(self.c[lastidx])
        i = i+1
        lastidx = lastidx - 1
        if lastidx < 0:
            lastidx = len(self.c)-1
```

Date:

Grant Thornton
An instinct for growth™

Rules:

1. Maintain 2 variables : One for size, One for index.

2. If increment in circular array's index, then mod the index with length of circular array [index = (index-1) % len(self

3. If decrement array check for the negative index, and set it to last index of array [lastindex = (self.start + self.size - 1) % len(self.e

Date:

Grant Thornton
An instinct for growth™

[Linear array - to circular array]

l = [10, 20, 30, 40, 50, 90] Start = 4
Size = 5

c = [0] * len(l)

```
Class CircularArray:
    def __init__(self, l, start, size):
        self.c = [0] * len(l)
        self.start = start
        self.size = size
        i = 0
        index = start
        while i < self.size:
            self.c[index] = l[i]
            i += 1
            index = (index+1) % len(self.c)
```

Date:

Grant Thornton
An instinct for growth™

STACK

Rules:

1) Write the operands in the output expression

2) Push the operators in the stacks only if it has a higher precedence than the top element of the stack. Otherwise keep poping the stack until the current operator can be pushed.

3) Push if opening bracket. Pop when closing bracket. (all operators till matching opening bracket)

Operator Precedence:

i) *, /, %

ii) +, -

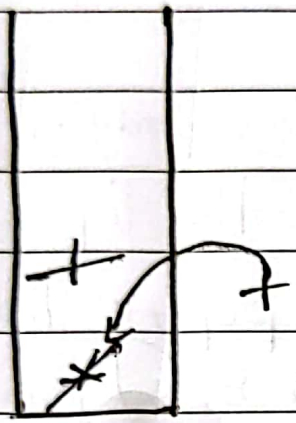
Date:

Grant Thornton
An instinct for growth™

Infix to Postfix

Infix: A*B+C

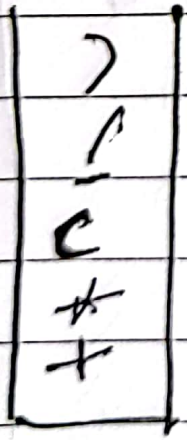
Postfix: AB*C+



Stack

"+" doesn't have the higest precedence "*">"+" so "*" will be popped.

Infix: A+B*(C+D/E)



Stack

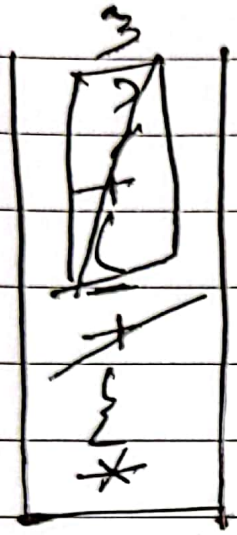
Postfix

ABCDE/+*+

Date:

Grant Thornton
An instinct for growth™

Infix: $A * \{B + C - (D + E / F)\}$



Stack

Postfix:

$ABC+DEF/+-*$

[দেখতে হবে Operator এর Precedence Stack এর মধ্যে যেটি আছে তার থেকে বড় নাকি? যদি বড় হয় তাহলে Stack এ রাখবে, Otherwise যদি ছোট বা তার সমান হয় তাহলে pop করতে হবে]

[Bracket shield এর মত কাজ করে, Bracket On হলে আগের Operator এর precedence matter করে না, Bracket close হলে Bracket এর ভিতরে থাকা সকল Operator pop হবে।]

Date:

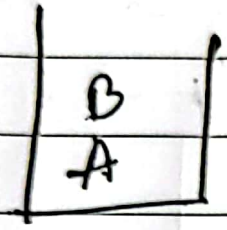
Postfix → Infix

Postfix

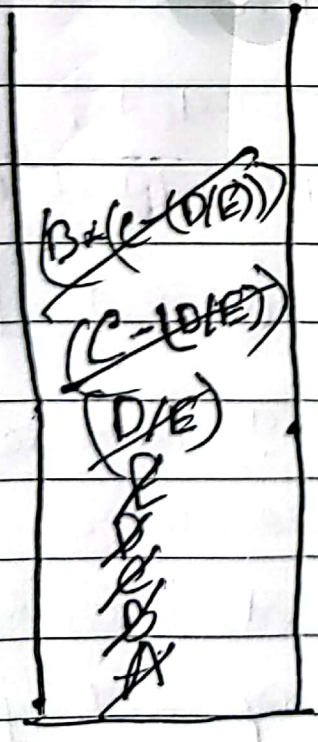
~~A B +~~

Infix

~~A + B~~



Postfix: ~~A~~ABCDE/-*+



Stack

~~Infix~~

~~(D/E)~~

~~(C-(D/E))~~

~~(B*(C-(D/E)))~~

(A+(B*(C-(D/E))))

Maintain করা

Date:

Grant Thornton
An instinct for growth™

↓ Mandatory

A Top item Operator এর ডানে থাকবে
2nd top item operator বামে থাকবে
Bracket বাদে দুইবার stack এ প্রবেশ করবে

B যতক্ষন পর্যন্ত Operator না আসে ততক্ষন পর্যন্ত Pop হতে থাকবে।

A Use a stack

A Push for operand

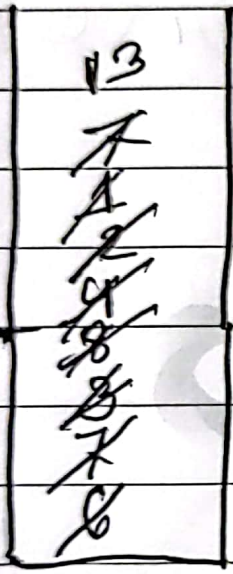
B Pop ~~first two elements for operator~~ from an equation and push it back in the stack.

Date:

Grant Thornton
An instinct for growth™

Postfix Evaluation

Postfix : 67384/-*+



Stack

⇒ 8/4 = 2

⇒ 3 - 2 = 1

⇒ 7 * 1 = 7

⇒ 6 + 7 = 13

* Answer will return in stack

Date:

Grant Thornton
An instinct for growth™

Stack Implementation (Array)

```
class Stack :
    stack = [ ]
    pointer = -1

    def push (self, element) :
        self.stack.append (element)
        self.pointer += 1

    def peek (self) :
        return (self.stack [self.pointer])

    def pop (self) :
        Value = self.stack [self.pointer]
        self.stack = self.stack [ :-1]
        self.pointer -= 1
        return Value
```

Date:

Grant Thornton
An instinct for growth™

Stack Implementation (Linked List)

```
Class stack:
    head = None

    def push (self, data):
        if self.head == None:
            self.head = Node(data)

        else:
            n = Node (data)
            n.ref = self.head
            self.head = n

    def peek (self):
        return (self.head.value)

    def pop (self):
        temp = self.head
        self.head = self.ref

        temp.value = None
        temp.ref = None
```