<pre>In [4]: class Node: definit(self,data): self.data=data self.next=None def printll(head): while head is not None: print(str(head.data)+">",end=" ") head=head.next print("None") def length(head): count=0 while head is not None: count=ount+1 head=head.next return count def insertion(head,data,i=0): if i<0 or i>length(head): return head curr=head prev=curr</pre>	
<pre>while head is not None: print(str(head.data)+">",end=" ") head=head.next print("None") def length(head): count=0 while head is not None: count=count+1 head=head.next return count def insertion(head,data,i=0): if i<0 or i>length(head): return head curr=head prev=None count=0 while count<i:< pre=""></i:<></pre>	
<pre>def length(head): count=0 while head is not None: count=count+1 head=head.next return count def insertion(head, data, i=0): if i<0 or i>length(head): return head curr=head prev=None count=0 while count<i:< pre=""></i:<></pre>	
<pre>count=count+1 head=head.next return count def insertion(head, data, i=0): if i<0 or i>length(head): return head curr=head prev=None count=0 while count<i:< pre=""></i:<></pre>	
<pre>if i<0 or i>length(head): return head curr=head prev=None count=0 while count<i:< pre=""></i:<></pre>	
<pre>curr=head prev=None count=0 while count<i:< pre=""></i:<></pre>	
curr=curr.next count=count+1	
<pre>if prev is None: newNode=Node(data) newNode.next=curr</pre>	
head=newNode else: newNode=Node(data) prev.next=newNode	
newNode.next=curr return head	
c1=Node(10) c2=Node(20)	
c3=Node(30) c4=Node(40) c1.next=c2	
<pre>c2.next=c3 c3.next=c4 printll(c1) print(length(c1))</pre>	
x=insertion(c1,200) printll(x) 10> 20> 30> 40> None	
4 200> 10> 20> 30> 40> None	
<pre>In [12]: class Node: definit(self, data): self.data=data self.next=None</pre>	
<pre>def printll(head): while head is not None: print(str(head.data)+">", end=" ")</pre>	
<pre>head=head.next print("None") def length(head): count=0</pre>	
<pre>while head is not None: count=count+1 head=head.next return count</pre>	
<pre>def insertion(head, data, i=0): if i<0 or i>length(head):</pre>	
<pre>return head curr=head prev=None count=0</pre>	
<pre>while count<i: curr="curr.next</pre" prev="curr"></i:></pre>	
<pre>count=count+1 newNode=Node(data) if prev is None: newNode.next=curr</pre>	
head=newNode else: prev.next=newNode newNode.next=curr	
<pre>return head def deletion(head, i=0):</pre>	
<pre>if i==0: head=head.next else: curr=head</pre>	
<pre>count=1 while count<i: curr="curr.next</pre"></i:></pre>	
count=count+1 curr.next=curr.next.next return head	
c1=Node(10) c2=Node(20) c3=Node(30) c4=Node(40)	
<pre>c1.next=c2 c2.next=c3 c3.next=c4</pre>	
<pre>printll(c1) print(length(c1)) x=insertion(c1,200,200) printll(x)</pre>	
<pre>y=deletion(x,length(x)-1) printll(y) 10> 20> 30> 40> None</pre>	
4 10> 20> 30> 40> None 10> 20> 30> None	
In []: #applications of Linkedlist 1.Used for implementing stacks and Queues 2.Dynamic memeory allocation	
3.Sparse Matrix #Real Life example: 1.Web browser hyper link 2.Image viewer	
3.Music Player 4.Youtube videos	
In []: #perform linear search in linkedlist> 10->20->30->40 key=40 return 3 else return -1 #nth node problem> 10->20->30->40 n=2>30 The first search in linear Parastructure that will follow the principle of LTFO(Last in first out)	
<pre>In []: #Stack> A linear Datastructure that will follow the principle of LIFO(Last in first out)> whatever is coming last moved out first In stack Insertion and Deletion both are done at the same end (Top) How we can implement to stack?</pre>	
1.Array 2.linkedlist Terminology of stack:	
<pre>push()> Insert any element into the stack pop()> delete any elemt from the stack peek()> return the top value</pre>	
<pre>In [19]: #Implement stack using array class Stack: definit(self):</pre>	
<pre>self.array=[] def push(self,item): self.array.append(item)</pre>	
<pre>self.array=[] def push(self,item): self.array.append(item) def pop(self): if len(self.array)==0: print("Stack is Empty!!!")</pre>	
<pre>self.array=[] def push(self,item): self.array.append(item) def pop(self): if len(self.array)==0: print("Stack is Empty!!!") return return self.array.pop() def peek(self): if len(self.array)==0:</pre>	
<pre>self.array=[] def push(self,item): self.array.append(item) def pop(self): if len(self.array)==0: print("Stack is Empty!!!") return return self.array.pop() def peek(self): if len(self.array)==0: print("Stack is Empty!!!") return return self.array]==0: print("Stack is Empty!!!") return return self.array[len(self.array)-1]</pre>	
<pre>self.array=[] def push(self,item): self.array.append(item) def pop(self): if len(self.array)==0: print("Stack is Empty!!!") return return self.array.pop() def peek(self): if len(self.array)==0: print("Stack is Empty!!!") return return self.array[len(self.array)-1] c=Stack() c.push(10) c.push(20) c.push(30)</pre>	
<pre>self.array=[] def push(self,item): self.array.append(item) def pop(self): if len(self.array)==0: print("Stack is Empty!!!") return return self.array.pop() def peek(self): if len(self.array)==0: print("Stack is Empty!!!") return return self.array[len(self.array)-1] c=Stack() c.push(10) c.push(20)</pre>	
<pre>self.array=[] def push(self,item): self.array.append(item) def pop(self): if len(self.array)==0: print("Stack is Empty!!!") return return self.array.pop() def peek(self): if len(self.array)==0: print("Stack is Empty!!!") return return self.array[len(self.array)-1] c=Stack() c.push(10) c.push(20) c.push(30) print(c.pop())</pre>	
<pre>self.array=[] def push(self.item): self.array.append(item) def pop(self): if len(self.array)==0: print("Stack is Empty!!") return return self.array.pop() def peek(self): if len(self.array)==0: print("Stack is Empty!!") return return self.array[len(self.array)-1] c=Stack() c.push(10) c.push(20) c.push(30) print("C.pop()) print(c.pop()) print(c.pop()) solution self.array Linkedlist count=0 In []: #Implementation of Stack using Linkedlist count=0</pre>	
<pre>self.array=[] def pusk(self,item): self.array.append(item) def pop(self): if len(self.array)=0: print("Stack is Empty!!!") return return self.array.pop() def peek(self): if len(self.array)=0: print("Stack is Empty!!!") return return return self.array[len(self.array)-1] c=Stack() c.push(20) c.push(20) c.push(20) print(c.pop()) print(c.pop()) print(c.pop()) print(c.pop()) #Implementation of Stack using Linkedlist count=0 for Push> i.Create a node 2.Make Connections 3.Change head</pre>	
<pre>self.array=[] def pup(self, item): self.array.append(item) def pop(self): if len(self.array)==0: print("Stack is Empty!!!") return return self.array.pop() def peek(self): if len(self.array)==0: print("Stack is Empty!!!") return return self.array[len(self.array)-1] c=Stack() c.push(10) c.push(20) c.push(20) c.push(30) print(c.pop()) print(c.pop()) print(c.pop()) #Implementation of Stack using Linkedlist count=0 for Push> 1.Create a node 2.Make Connections</pre>	
<pre>self.array=[] def push(self,item): self.array.append(item) def pop(self): if len(self.array)==0: print("stack is Empty!!!") return return self.array.pen() def peek(self): if len(self.array)==0:</pre>	
<pre>self.array=[] def push(self.item): self.array.append(item) def pop(self): if len(self.array)==0: print("stack is Empty!!!") return return self.array.pep() def peek(self): if len(self.array)==0: print("stack is Empty!!!") return return self.array ==0: print("stack is Empty!!!") return return return self.array ==0: print("stack is Empty!!!") return return</pre>	
<pre>self.array=[] def push(self, item): self.array.append(item) def pop(self): if len(self.array)==0: print("Stack is Empty!!") return return self.array.pop() def peed: if len(self.array)==0:</pre>	
<pre>self.array=[] def push(self.item): self.array.append(item) def pofusEdf: if lon(self.array)=0:</pre>	
<pre>self.array=[] def pushs[stritem]: self.array.appen(item) def pop(self): if len(self.array.appen(item) return self.array.pop() def peck(self): return self.array.pop() def peck(self): return self.array.pop() def peck(self): return self.array.pop() def peck(self): return self.array[len(self.array)-1] return self.array[len(self.array)-1] c=Scack() c.push(18) c.push(28) c.push(28) print(c.pop()) print(c.pop()) print(c.pop()) print(c.pop()) print(c.pop()) print(c.pop()) return self.array.elen(self.array)-1 c=Scack() return self.array[len(self.array)-1] c=Scack() print(c.pop()) print(c.pop()) print(c.pop()) print(c.pop()) print(c.pop()) return self.array[len(self.array)-1] for push self.array[len(self.array]-1] for pop l.nead=head.next 2.count=count-1 for peek i.return head.data for checking stack is empty or not if count=self.</pre>	
<pre>self.array=[] def posh(self,array.append(iten) self.array.append(iten) def pos(self): if len(self.array)=0: print("Stack is Empty!!") return retur</pre>	
<pre>self.array=[] def puns(self.item); self.array.append(item) def pos(self); if len(self.array)=mai: print("Stock is Empty!!") return self.array.ppt() def peek(self); if len(self.array)=mai: print("Stock is Empty!!") return self.array.ppt() def peek(self); if len(self.array)=mai: print("stock is Empty!!") restack() c.publ(self.array)=mai: c.publ(self.array)=mai: c.publ(self.array)=mai: c.publ(self.array)=mai: print(c.publ(self.array)=1) print(c.publ(self.array)=1)</pre>	
<pre>asil.array=[def postset].tem) set.array.apend(item) def postset]: if len(sel.array)==0: print('Stack is Empty!!") return sel.array.pop() def peek(sel?): if len(sel.array)==0:</pre>	
self array=[
self array=[der post(self, trm) self array=grend(ton) der post(self, trm) der post(self, trm) if lest(self, trm) return melf array=ms; return return melf. array[ten(self.array)-1] c.push(28) c.push(28) c.push(28) c.push(28) c.push(28) c.push(28) prant(c.pos()) prant(c.pos()) prant(c.pos()) prant(c.pos()) prant(c.pos()) prant(c.pos()) prant(c.pos()) prant(c.pos()) return self array=ms; return melf array ten(self.self.self.self.self.self.self.self.	