

## LINKED LIST:-

A linked list consists of nodes where each node contains a data element and the reference (link) to the next node in the list. First node of linked list is called "head". Last node of linked list is called "tail".



# => IMPLEMENTING STACKS THROUGH

## LINKED LIST

# First making a class node

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

→ In order to create nodes in a linked list we first create a class node with attribute data

# Now making a class stack

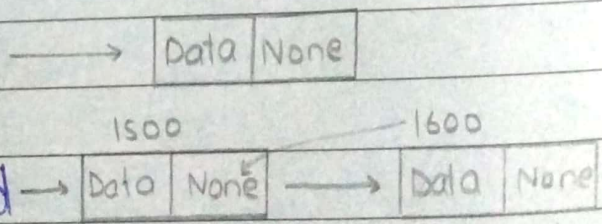
```
class LinkStack():  
    def __init__(self):  
        self.head = None  
        self.tail = None
```

→ Now in order to implement stacks in linked list we now creating a class stack and initializing head and tail to None

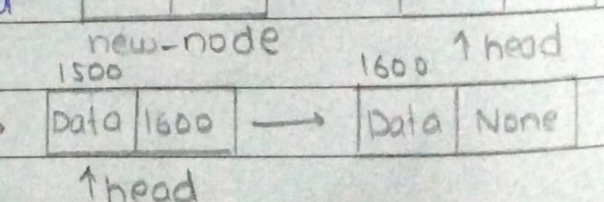
# Now writing a method of push

```
def push(self, data):  
    new_node = node(data)
```

new\_node.next = self.head



self.head = new\_node



\* head is always the first node of linked list

\* In push function we create nodes at head



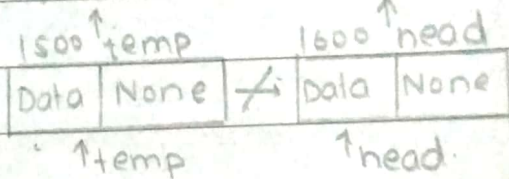
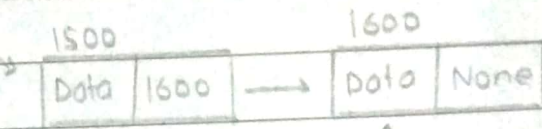
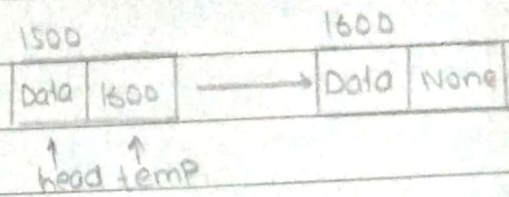
# Now writing a function of pop

```
def pop(self):
```

```
    temp = self.head
```

```
    self.head = temp.next
```

```
    temp.next = None
```



=> In pop function we delete the first node by disconnecting its link to second node.

# Now writing a function of show-all

```
def show-all(self):
```

```
    temp = self.head
```

```
    If self.head is None:
```

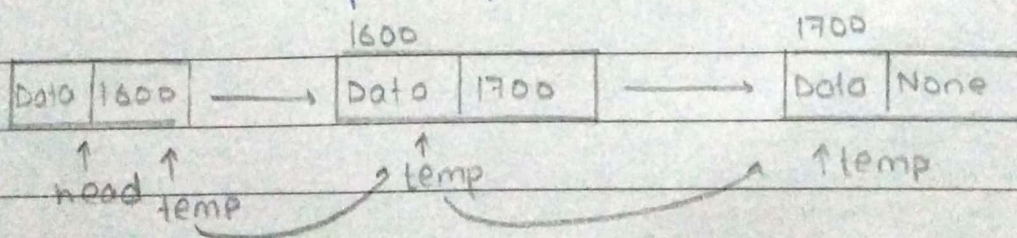
```
        Print ("List is Empty")
```

```
    return
```

```
    While temp:
```

```
        Print (temp.data)
```

```
        temp = temp.next
```



# In order to print our linked list we first make a temporary variable equal to head and then traverse it to last.



# => IMPLEMENTING QUEUES THROUGH

## LINKED LIST:-

# First making a class node

```
class node():
```

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

```
        self.data = data
```

```
        self.next = None
```

→ First we make a class node in order to create nodes in a linked list i.e. with attribute data

1500	Data	None
------	------	------

# Now making a class queue

```
class Queue():
```

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

```
        self.head = None
```

```
        self.tail = None
```

→ In order to implement queue through Linked list we now creating a class queue and initializing head & tail to None

# Now writing a method of enqueue

```
def enqueue(self, data):
```

If self.head is None:

```
    new_node = node(data)
```

```
    self.head = new_node
```

```
    self.tail = new_node
```

for first node

1500	Data	None
------	------	------

1500	Data	None
------	------	------

↑ head ↑ tail

else:

```
    new_node = node(data)
```

1500	Data	None
------	------	------

↑ head ↑ tail

1600	Data	None
------	------	------

new\_node

```
    self.tail.next = new_node
```

1500	Data	1600
------	------	------

↑ head ↑ tail

1600	Data	None
------	------	------



self.tail = self.tail.next → 

Data	1600
------	------

 → 

Data	None
------	------

  
↑ head ↑ tail

\* Last node of linked list is "tail"

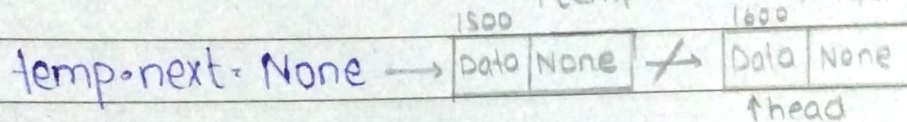
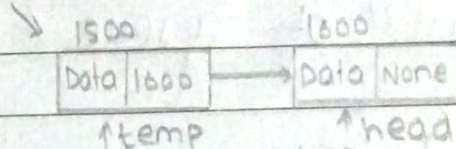
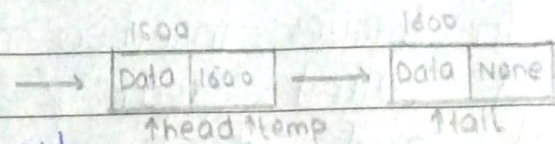
\* In enqueue function we add nodes at tail.

# Now writing a method of dequeue

def dequeue(self):

temp = self.head

self.head = temp.next



# Now writing a method of show-all

def show-all(self):

temp = self.head

If self.head is None:

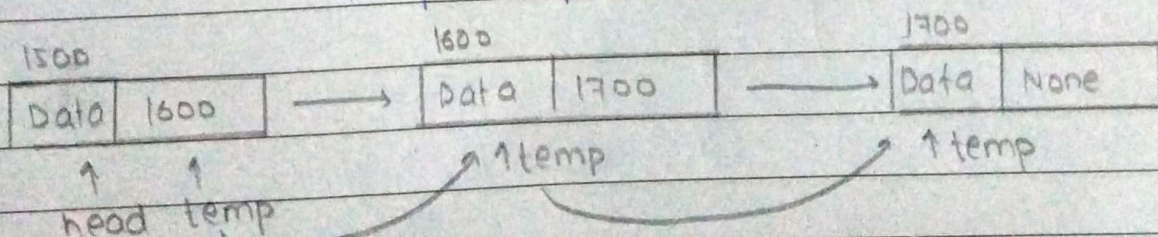
Print("List is Empty")

Return

While temp:

Print(temp.data)

temp = temp.next



⇒ In order to print whole linked list we need to traverse whole linked list.



## CONCLUSION:-

=> In stacks both insertion and deletion can be performed at one end called "top" of the stack and "head" of the linked list.

=> In Queues insertion can be performed at one end called "rear" / "tail" of the linked list and deletion can be performed at "front" of queue / "head" of linked list.