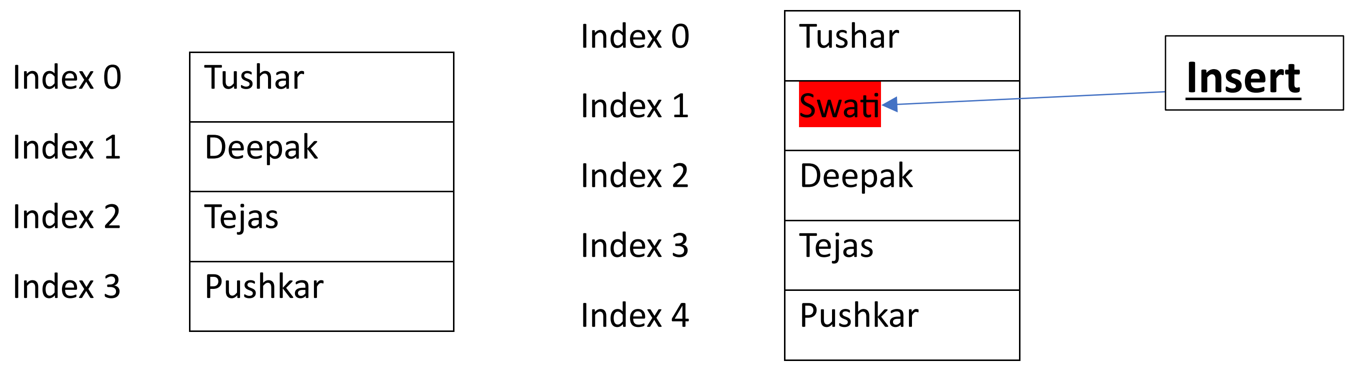
<u>Linked List</u>

We will be implementing linked list in python and will go through various operations on the linked list.

Suppose I have an array of student names and it this is a memory layout of those students names.

names=['Tushar','Deepak','Tejas','Pushkar']

There are like the fire elements in this array if you want to insert an element 'Swati ' at location number 1 then it will insert 'Swati' at location number 1 and it will swap all the elements .So it will copy Deepak from location number to 2,Tejas from location number 2 to 3 and so on.



Using array or list this is possible by using insert function like that names.insert(1,’Swati’)

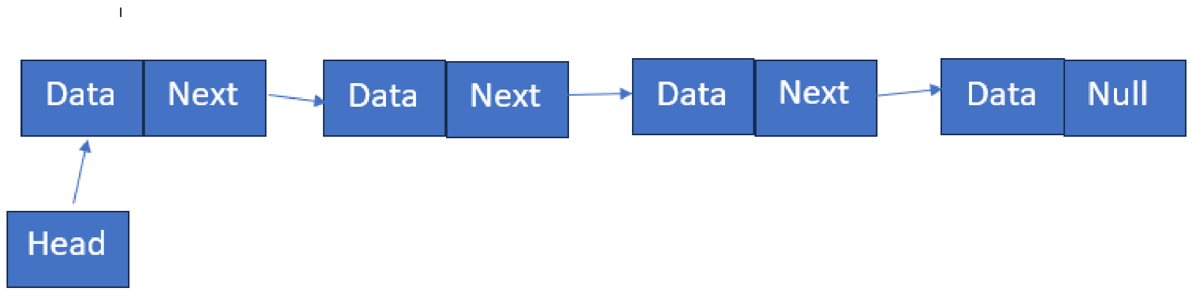
Now let us discuss this data structure with individual values are stored with two different areas of memory. That structure is called linked list.

0x1100	Tushar
0x1104	Deepak
0x1108	Tejas
0x1112	Pushkar
0x1114	Sarika

A linked list is a sequence of data elements, which are connected together via links. Each data element contains a connection to another data element in form of a pointer.

A Linked list is made up of nodes, where a node is made up of two parts:

1. **Data Part**: Holds the data
2. **Address Part**: Holds the address of the next node.



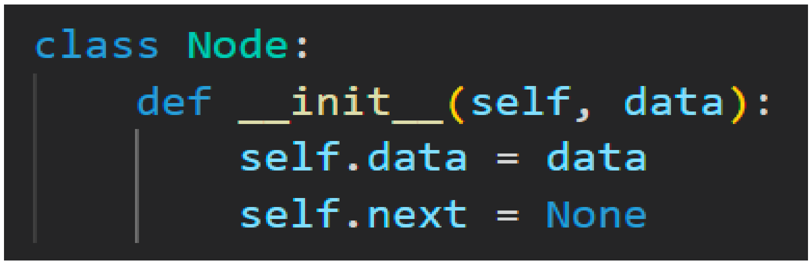
As you can see, every node has some data stored in it, and it also stores the location of the next node. It is not necessary that all the nodes get saved in contiguous memory locations, next to each other, that is why we need to store location of the next node in the previous node to be able to access it like its a chain or list.

Head is a pointer which always points to the first node of the list, if Head points to nothing, it means the linked list is empty.

Our first node is where **head** points and we can access all the elements of the linked list using the **head.**

Creating a Node Class

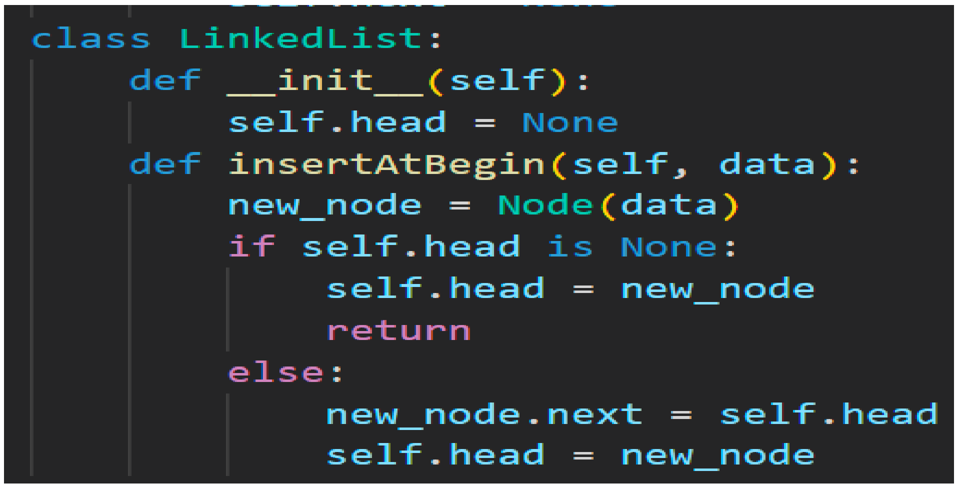
We have created a Node class in which we have defined a __init__ function to initialize the node with the data passed as an argument and a reference with None because if we have only one node then there is nothing in its reference.



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

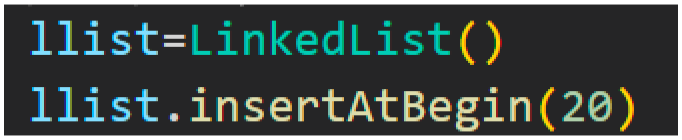
Insertion at Beginning in Linked List

This method inserts the node at the beginning of the linked list. In this method, we create a **new_node** with the given **data** and check if the head is an empty node or not if the head is empty then we make the **new_node** as **head** and **return** else we insert the head at the next **new_node** and make the **head** equal to **new_node.**



```python
class LinkedList:
    def __init__(self):
        self.head = None
    def insertAtBegin(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
            return
        else:
            new_node.next = self.head
            self.head = new_node
```

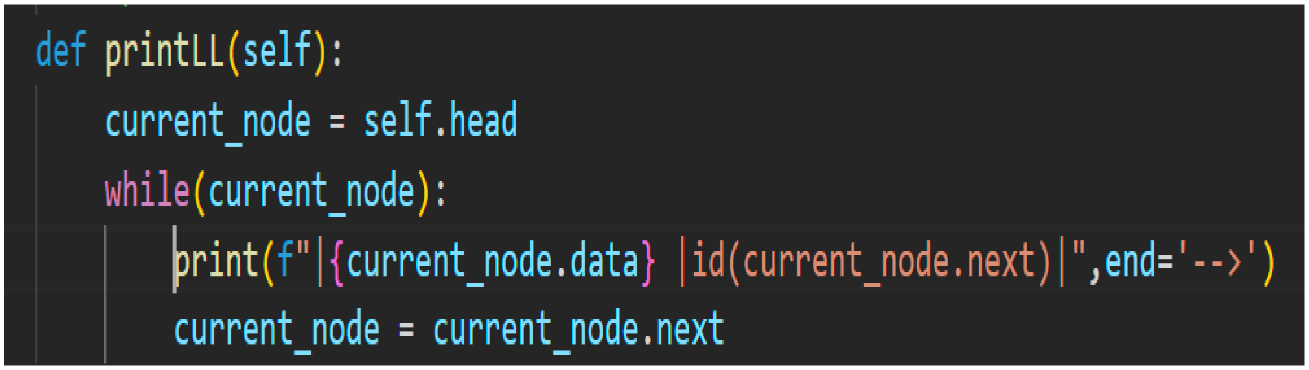
Called a function **insertAtBegin()** using object of LinkedList class:



```
llist=LinkedList()
llist.insertAtBegin(20)
```

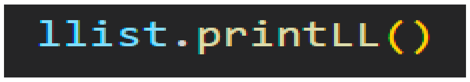
Linked List Traversal in Python

This method traverses the linked list and prints the data of each node. In this method, we made a **current_node** equal to the **head** and iterate through the linked list using a **while loop** until the **current_node** become None and print the data of **current_node** in each iteration and make the **current_node** next to it.



```
def printLL(self):
    current_node = self.head
    while(current_node):
        print(f"|{current_node.data} |id(current_node.next)|",end='-->')
        current_node = current_node.next
```

Called a function **printLL()** using object of LinkedList class:

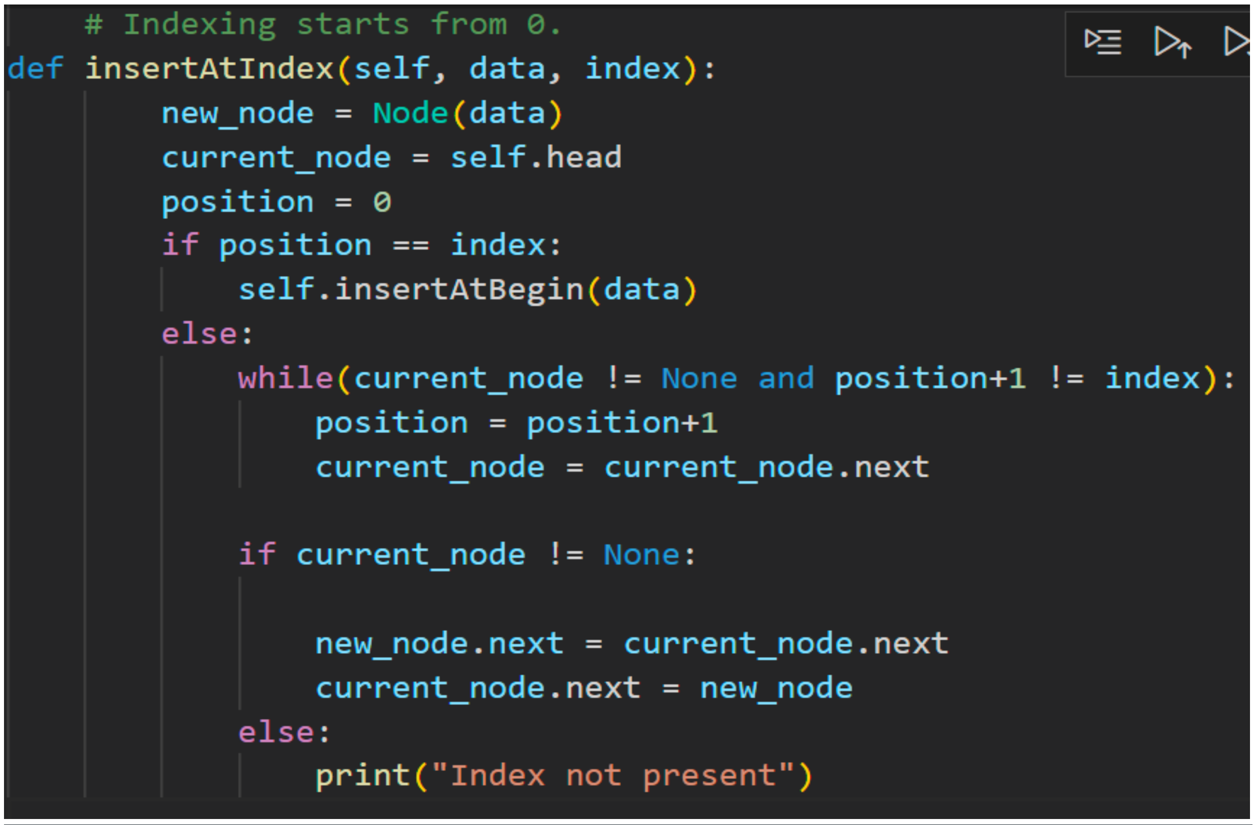


```
llist.printLL()
```

Output of program:

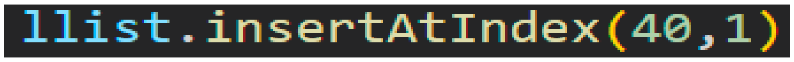
```
|20 |id(current_node.next)|-->
```

Insert a Node at a Specific Position in a Linked List:



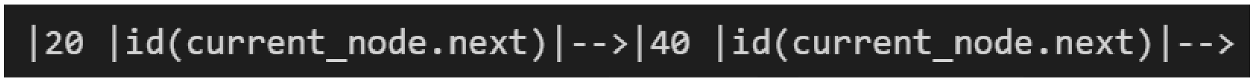
```
    # Indexing starts from 0.
def insertAtIndex(self, data, index):
        new_node = Node(data)
        current_node = self.head
        position = 0
        if position == index:
            self.insertAtBegin(data)
        else:
            while(current_node != None and position+1 != index):
                position = position+1
                current_node = current_node.next

            if current_node != None:

                new_node.next = current_node.next
                current_node.next = new_node
            else:
                print("Index not present")
```

Called a function **insertAtIndex()**using object of LinkedList class:



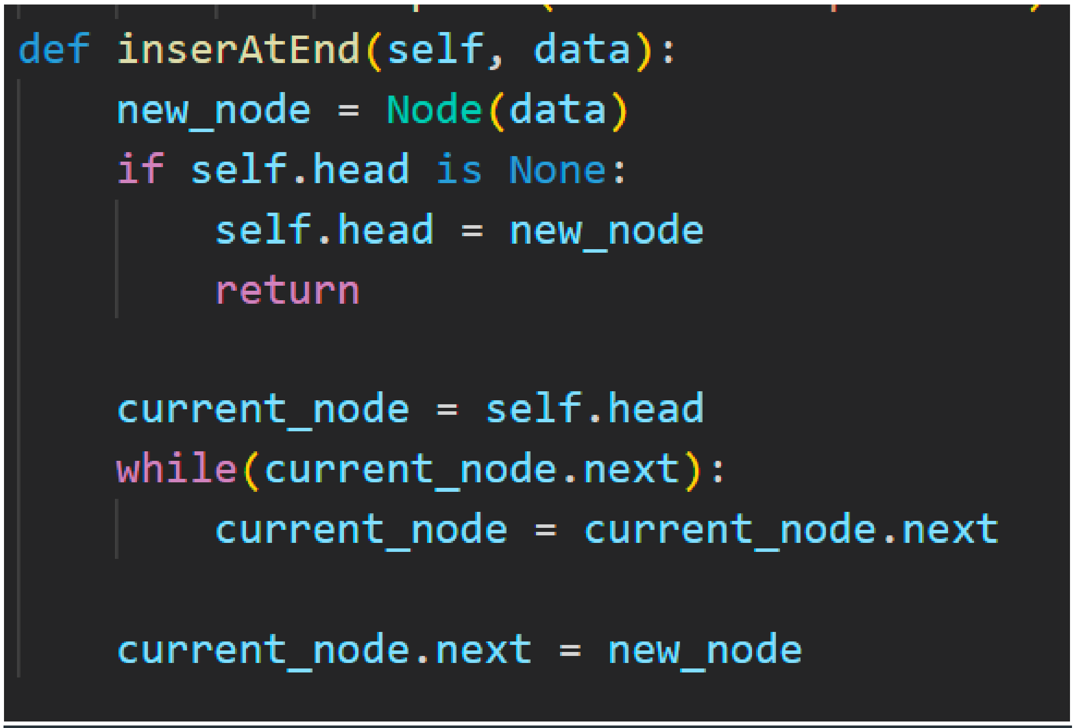
```
llist.insertAtIndex(40,1)
```

<u>Output of program:</u>



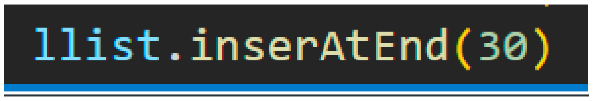
```
|20 |id(current_node.next)|-->|40 |id(current_node.next)|-->
```

<u>Insertion in Linked List at End:</u>



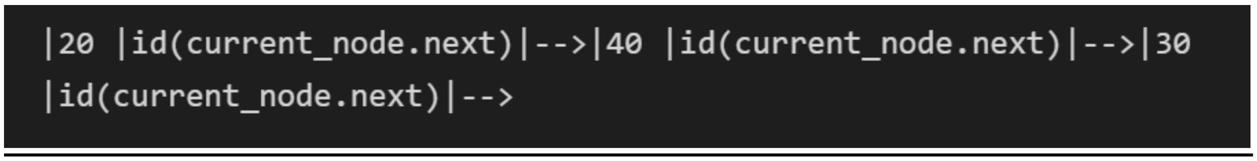
```
def inserAtEnd(self, data):
    new_node = Node(data)
    if self.head is None:
        self.head = new_node
        return

    current_node = self.head
    while(current_node.next):
        current_node = current_node.next

    current_node.next = new_node
```

Called a function **insertAtEnd()**using object of LinkedList class:



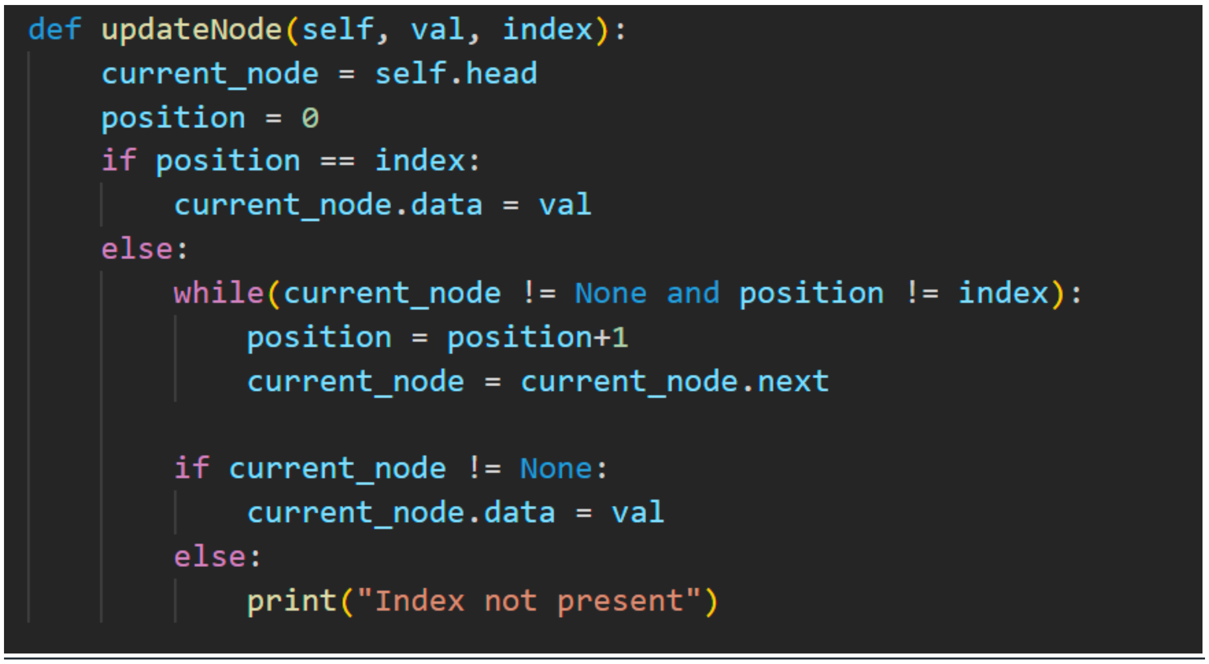
```
llist.inserAtEnd(30)
```

Output of program:



```
|20 |id(current_node.next)|-->|40 |id(current_node.next)|-->|30
|id(current_node.next)|-->
```

Update the Node of a Linked List:



```python
def updateNode(self, val, index):
    current_node = self.head
    position = 0
    if position == index:
        current_node.data = val
    else:
        while(current_node != None and position != index):
            position = position+1
            current_node = current_node.next

        if current_node != None:
            current_node.data = val
        else:
            print("Index not present")
```

Called a function **updateNode()**using object of LinkedList class:

```
llist.updateNode(50,1)
```

Output of program:

```
|20 |id(current_node.next)|-->|50 |id(current_node.next)|-->|30
|id(current_node.next)|-->
```

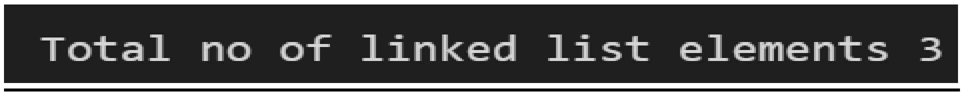
Get Length of a Linked List in Python:

```
def sizeOfLL(self):
    size = 0
    if(self.head):
        current_node = self.head
        while(current_node):
            size = size+1
            current_node = current_node.next
        return size
    else:
        return 0
```

Called a function **sizeOfLL()**using object of LinkedList class:

```
size=llist.sizeOfLL()
print("Total no of linked list elements",size)
```

Output of program:



```
Total no of linked list elements 3
```

Just like insertion an element into a list ,we can remove an element from first,last and particular index position.

Advantages of a Linked List

1. Insertion and Deletion are very easy, in case of Linked List, becasue all we have to do is update next pointer.
2. We can easily implement Stack and Queue data structures using Linked List.

Disadvantages of a Linked List:

1. Random access is not possible, to access any node we must traverse the Linked List. That is why it is said, if you want to carry out insertion and updation on a set of data use Linked List, but if you want to traverse and search for data, array is better.