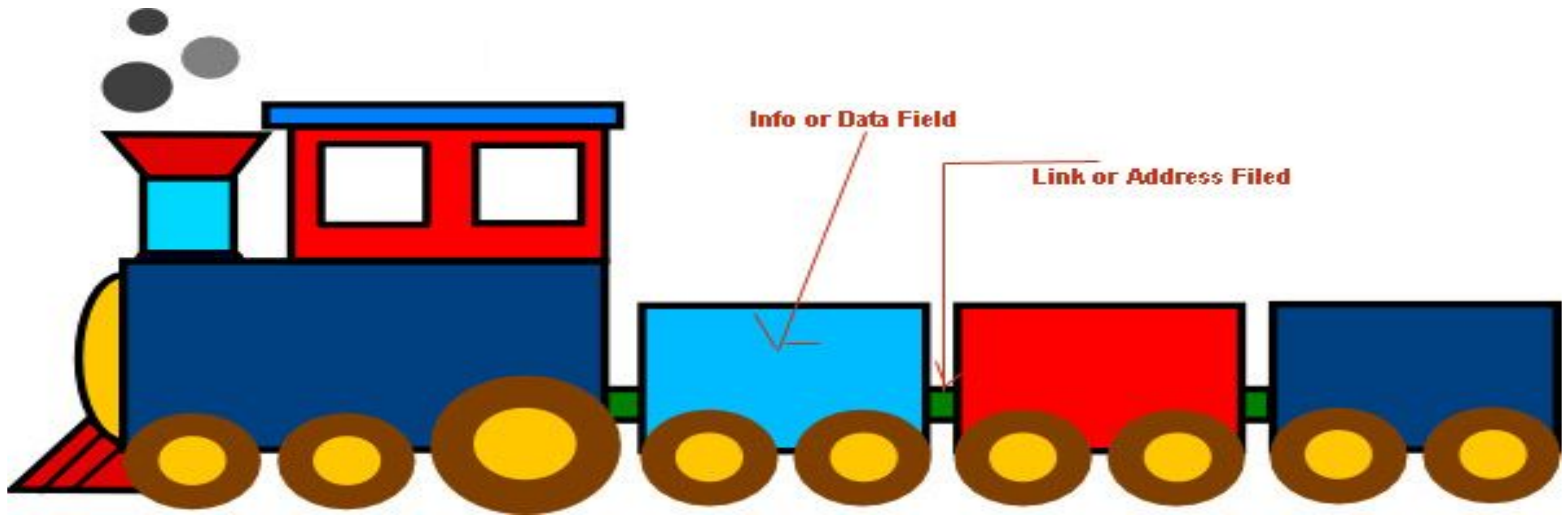
Linked List

- Linked List can be defined as collection of objects called nodes that are randomly stored in the memory.
- A node contains two fields i.e. data stored at that particular address and the pointer which contains the address of the next node in the memory.
- The last node of the list contains pointer to the null.

Linked List

- A linked list is a linear data structure.
- Nodes make up linked lists.
- Nodes are structures made up of data and a pointer to another node.
- Usually the pointer is called next.



Introduction to Linked List

Linked List

- **The elements of a linked list are not stored in adjacent memory locations as in arrays.**

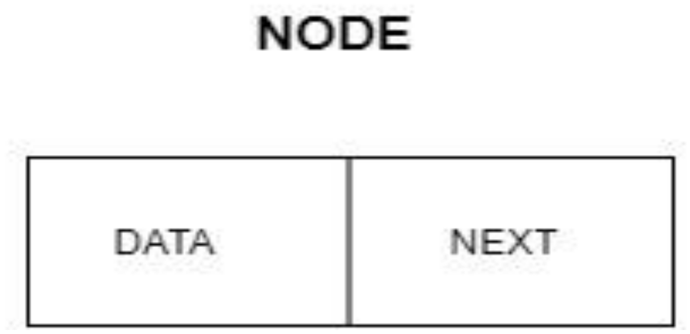
- **It is a linear collection of data elements, called nodes.**

Continued….

Linked List

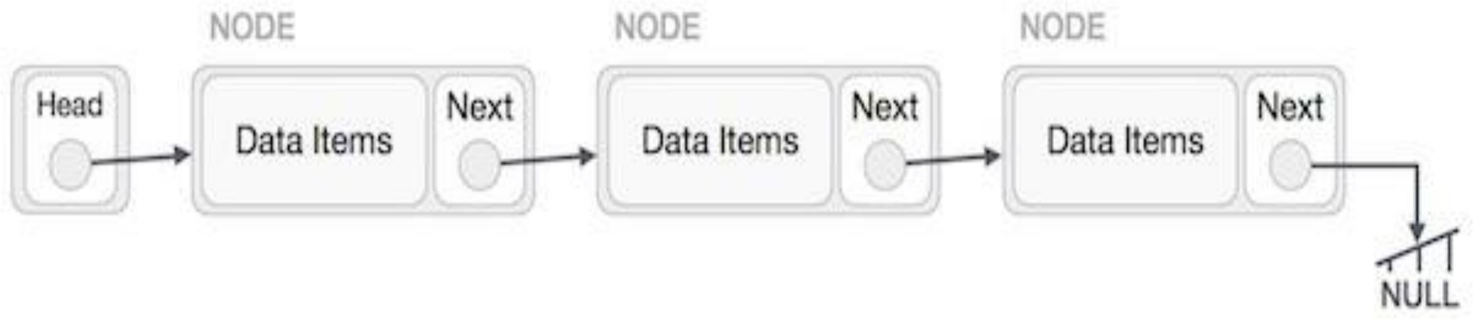
- In a linear or single-linked list, a node is connected to the next node by a single link.

- A node in this type of linked list contains two types of fields
  - data: which holds a list element
  - next: which stores a link (i.e. pointer) to the next node in the list.



Linked List Representation

- Linked list can be visualized as a chain of nodes, where every node points to the next node.



- As per the above illustration, following are the important points to be considered.
  - Linked List contains a link element called first(Head).
  - Each link carries a data field(s) and a link field called next.
  - Each link is linked with its next link using its next link.
  - Last link carries a link as null to mark the end of the list.

Properties of linked list

- The nodes in a linked list are not stored contiguously in the memory
- You don’t have to shift any element in the list
- Memory for each node can be allocated dynamically whenever the need arises.
- The size of a linked list can grow or shrink dynamically

Basic Operations on Linked List

- Following are the basic operations supported by a list.
  - **Insertion** − Adds an element in the list.
  - **Deletion** − Deletes an element from the list.
  - **Display** − Displays the complete list.
  - **Search** − Search an element using the given key.
  - **Delete** − Delete an element using the given key.

Arrays & Linked list

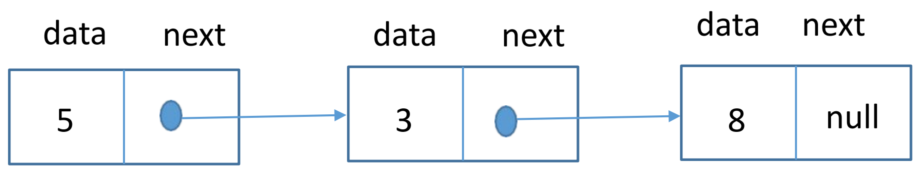
Arrays	**Linked list**
Fixed size: Resizing is expensive	Dynamic size
Insertions and Deletions are inefficient: Elements are usually shifted	Insertions and Deletions are efficient: No shifting
Random access i.e., efficient indexing	No random access Not suitable for operations requiring accessing elements by index such as sorting
No memory waste if the array is full or almost full; otherwise may result in much memory waste.	Since memory is allocated dynamically(acc. to our need) there is no waste of memory.
Sequential access is faster [Reason: Elements in contiguous memory locations]	Sequential access is slow [Reason: Elements not in contiguous memory locations]

Types of Link List

- Following are the various types of linked list.
  - **Singly Linked List** − Item navigation is forward only.
  - **Doubly Linked List** − Items can be navigated forward and backward.
  - **Circular Linked List** − Last item contains link of the first element as next
  - **Circular Doubly Linked List** − Last item contains link of the first element as next and the first element has a link to the last element as previous. Items can be navigated forward and backward.

Singly Linked list

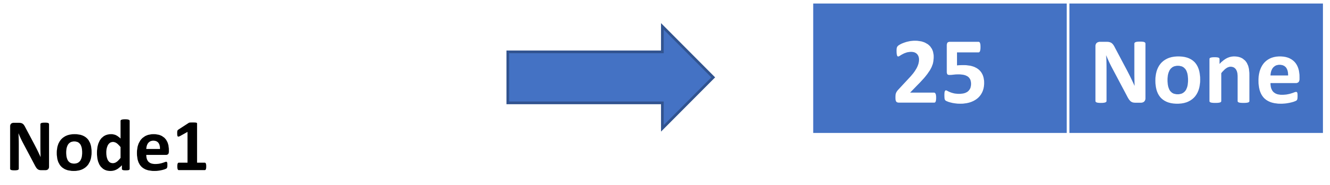
- A singly linked list is a dynamic data structure which may grow or shrink, and growing and shrinking depends on the operation made.
- In this type of linked list each node contains two fields one is data field which is used to store the data items and another is next field that is used to point the next node in the list.



Creating a node of linked list

```
# Node class (Creating a node of linked list)
class Node:
      # Function to initialize the node object
    def __init__(self, data):
        self.data = data  # Assign data
        self.next = None  # Initialize next as null
```

Node1=Node(25)



Creating an empty linked list

```
# Node class (Creating a node of linked list)
class Node:
      # Function to initialize the node object
   def __init__(self, data):
      self.data = data  # Assign data
      self.next = None  # Initialize next as null

# Linked List class (Linking the nodes of linked list)
class LinkedList:
      # Function to initialize the Linked List object
   def __init__(self):
      self.head = None
```

- init is a special method which is also called class constructor
- init method is called whenever new instance of a class is created and used to initialize the fields of a node
- Each node contain 2 fields.
- Using node class we are creating individual node not linking them

Creating a linked list with single node

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

class LinkedList:
    def __init__(self):
    self.head = None

LL = LinkedList()
LL.head = Node(3)
print(LL.head.data)
```

Creation and Traversal of single linked list

```
# A single node of a singly linked list
class Node:
def __init__(self, data):
  self.data = data
  self.next = None
```

```
# A Linked List class with a single
  head node
class LinkedList:
 def __init__(self):
  self.head = None
```

```
# insertion method for the linked list
 def insert(self, data):
  newNode = Node(data)
  if(self.head):
   current = self.head
   while(current.next):
    current = current.next
   current.next = newNode
  else:
   self.head = newNode
```

Creation and Traversal of single linked list (contd..)

```
# print method for the linked list
 def printLL(self):
   current = self.head
   while(current):
     print(current.data)
     current = current.next

# Singly Linked List with insertion and print methods
LL = LinkedList()
LL.insert(3)
LL.insert(4)
LL.insert(5)
LL.printLL()
```

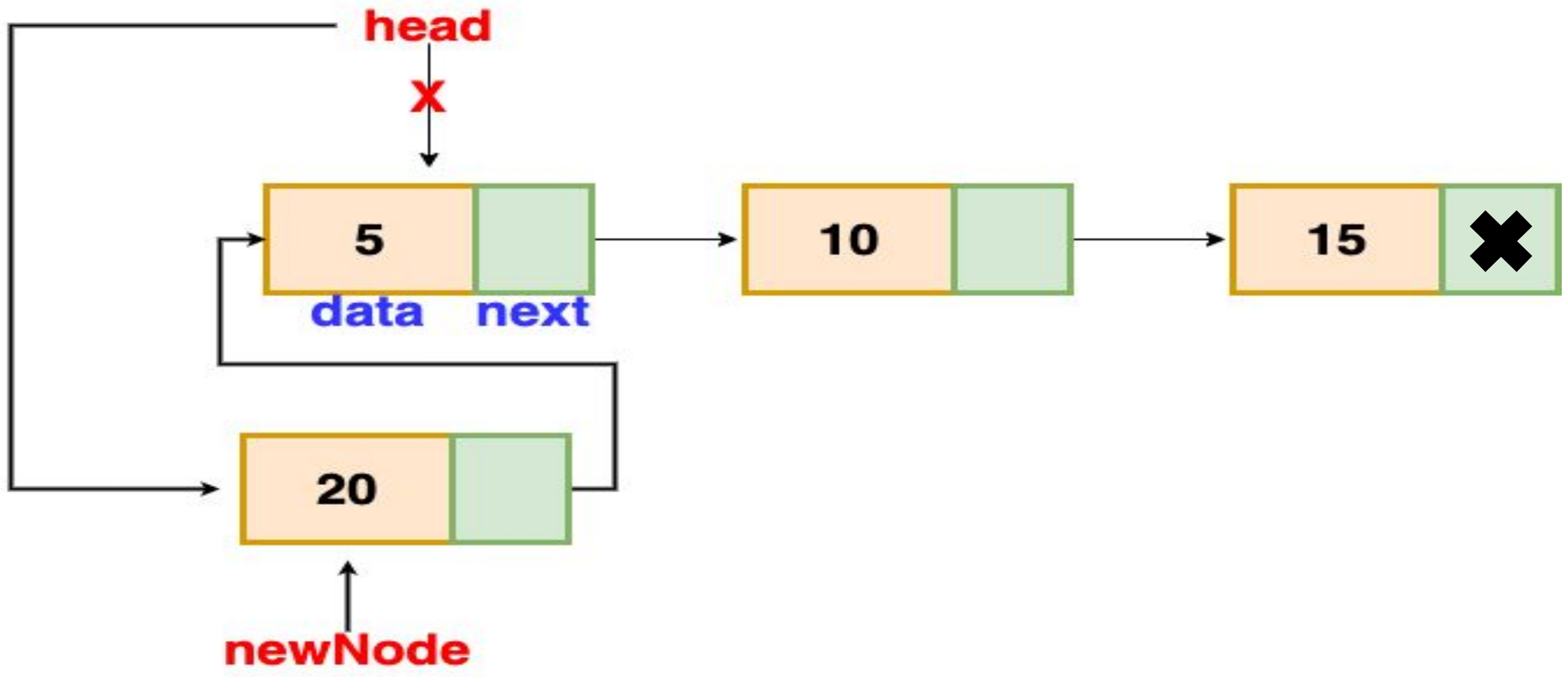
Insertion in a Single Linked List

- There are three possible positions where we can enter a new node in a linked list –
    - **Insertion at beginning**
    - **Insertion at end**
    - **Insertion at given position**

- Adding a new node in linked list is a more than one step activity.

Insertion in a Single Linked List (at beginning)

- Insertion at beginning



Insertion at the beginning

Insertion in single linked list (at beginning)

```
# A single node of a singly linked list
class Node:
def __init__(self, data):
    self.data = data
    self.next = None
```

```
# A Linked List class with a single
    head node
class LinkedList:
  def __init__(self):
    self.head = None
```

```
# insertion method for the linked list at
    beginning

def insert_beg(self, data):
    newNode = Node(data)
    if(self.head):
      newNode.next=self.head
      self.head=newNode
    else:
      self.head = newNode
```

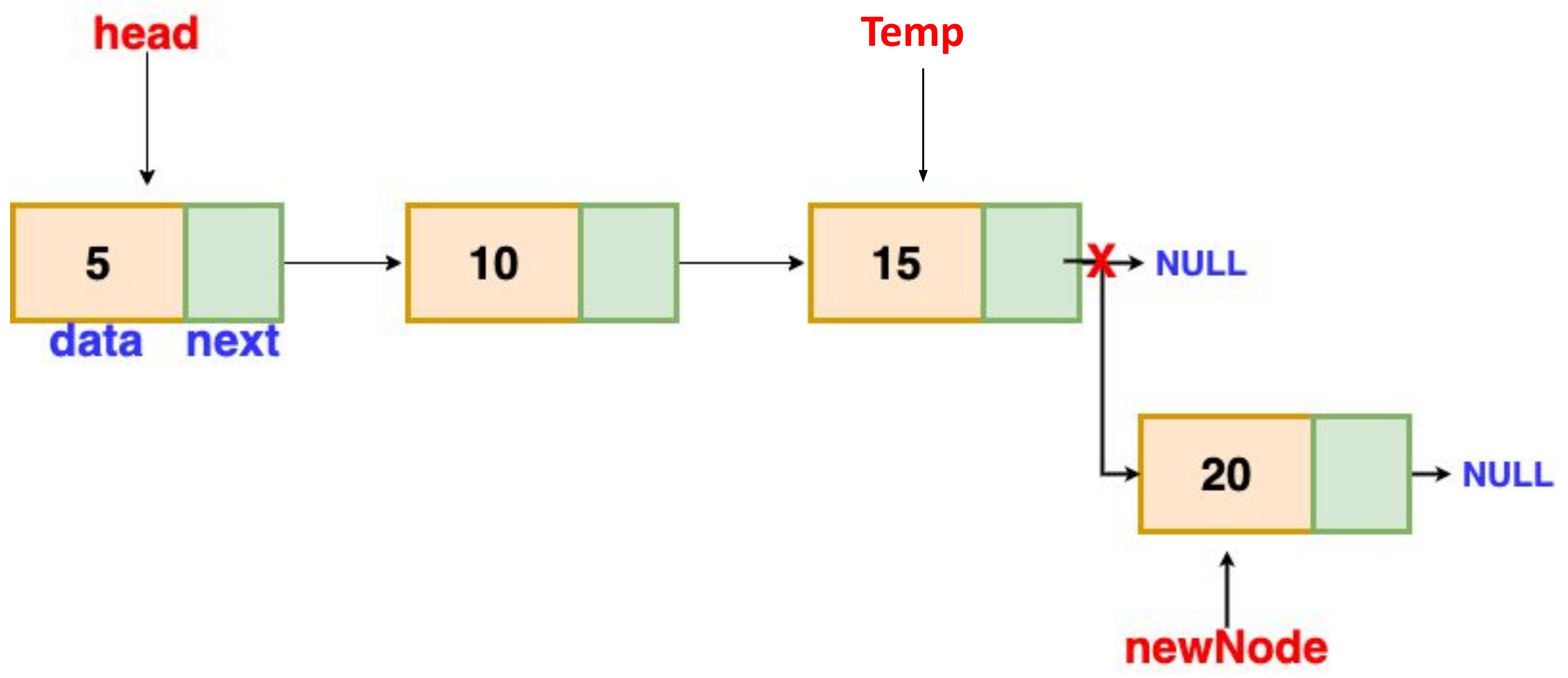
Insertion in single linked list (at beginning) (contd..)

```
# print method for the linked list
  def printLL(self):
    current = self.head
    if(current!=None):
        print("The List Contains:",end="\n")
        while(current):
            print(current.data)
            current = current.next
    else:
        print("List is Empty.")

# Singly Linked List with insertion and print methods
LL = LinkedList()
LL.insert_beg(3)
LL.insert_beg(4)
LL.insert_beg(5)
LL.printLL()
```

Insertion in a Single Linked List (at end)

- **Insertion at end**



Insertion at the end

Insertion in single linked list (at end)

```
# A single node of a singly linked list
class Node:
def __init__(self, data):
    self.data = data
    self.next = None
```

```
# A Linked List class with a single
    head node
class LinkedList:
  def __init__(self):
    self.head = None
```

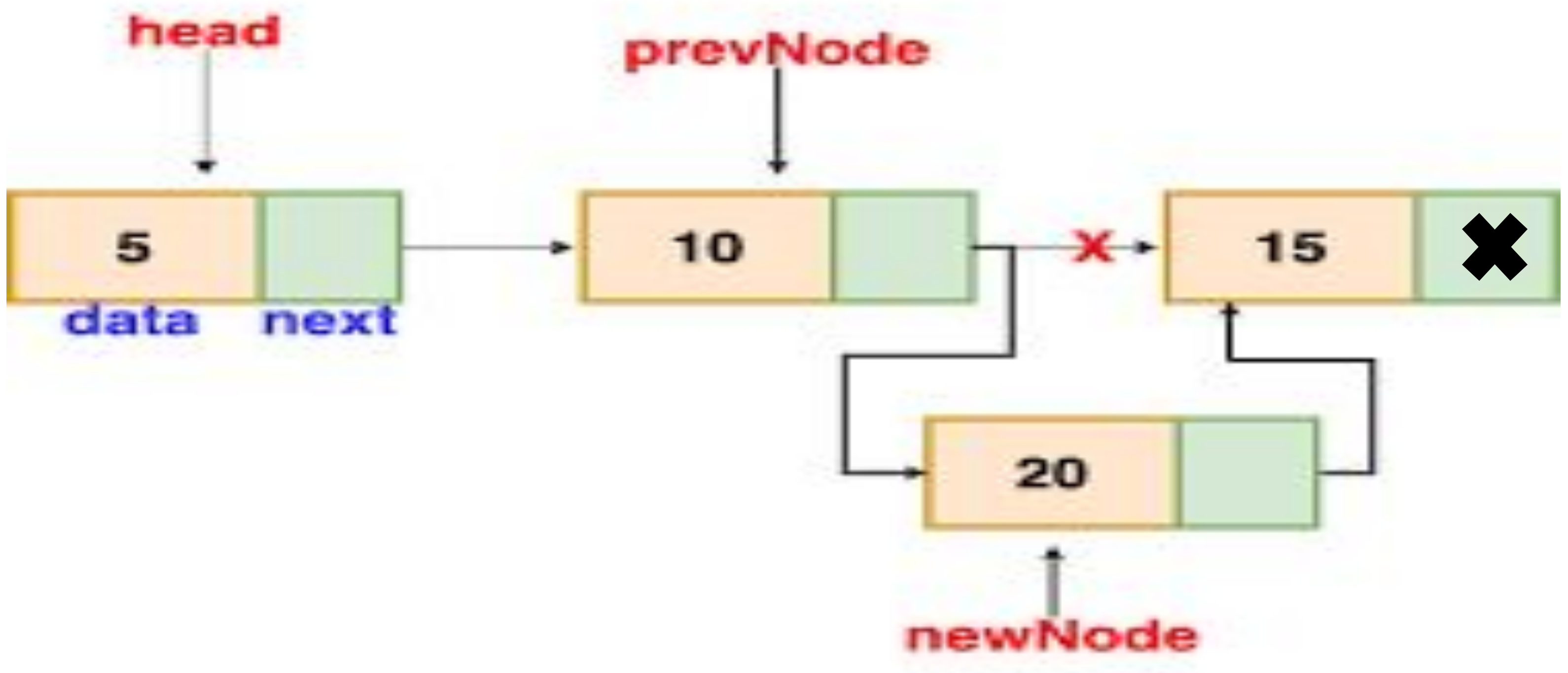
```
# insertion method for the linked list
    at end
 def insert_end(self, data):
   newNode = Node(data)
   if(self.head):
     current = self.head
     while(current.next):
       current = current.next
     current.next = newNode
   else:
     self.head = newNode
```

Insertion in single linked list (at end) (contd..)

```
# print method for the linked list
 def printLL(self):
  current = self.head
  if(current!=None):
    print("The List Contains:",end="\n")
    while(current):
      print(current.data)
      current = current.next
  else:
    print("List is Empty.")
# Singly Linked List with insertion and print methods
LL = LinkedList()
LL.insert_end(3)
LL.insert_end(4)
LL.insert_end(5)
LL.printLL()
```

Insertion in a Single Linked List (at given position)

- **Insertion at given position**



Insertion after a given node

Insertion in single linked list (at position)

```
# A single node of a singly linked list
class Node:
def __init__(self, data):
    self.data = data
    self.next = None


# A Linked List class with a single
    head node
class LinkedList:
  def __init__(self):
    self.head = None
```

```
# creation method for the linked list
  def create(self, data):
    newNode = Node(data)
    if(self.head):
      current = self.head
      while(current.next):
        current = current.next
      current.next = newNode
    else:
      self.head = newNode
```

Insertion in single linked list (at position)

```
# insertion method for the linked list at given position

def insert_position(self, data, pos):
  newNode = Node(data)
  if(pos<1):
    print("\nPosition should be >=1.")

  elif(pos==1):
    newNode.next=self.head
    self.head=newNode
else:
  current=self.head
  for i in range(1, pos-1):
    if(current!=None):
      current=current.next
  if(current!=None):
    newNode.next=current.next
    current.next=newNode
  else:
    print("\nThe previous node is null.")
```

Insertion in single linked list (at position)

```
# print method for the linked list
 def printLL(self):
   current = self.head
   if(current!=None):
     print("The List 
   Contains:",end="\n")
     while(current):
       print(current.data)
       current = current.next
   else:
     print("List is Empty.")
```

```
# Singly Linked List with insertion and
   print methods
LL = LinkedList()
LL.create(2)
LL.create(3)
LL.create(4)
LL.create(5)
LL.create(6)
LL.insert_position(9, 4)
LL.printLL()
```