Data Structure with Python
Linked List

MR. V. M. Vasava

GPG,IT Dept. Surat

Agenda

- Introduction about LL
- Types of Linked List
- Operations of Linked List

Introduction about LL

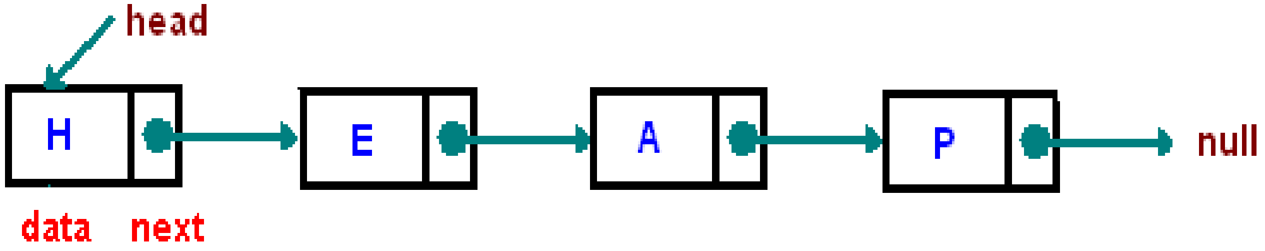
- **Linked list** elements are not stored at contiguous location but the elements are **linked** using pointers.
- A linked list is a linear data structure where each element is a separate object.
- A *linked list* is a series of connected *nodes*
- Each node contains at least

1.A piece of data (any type)

2.Pointer to the next node in the list

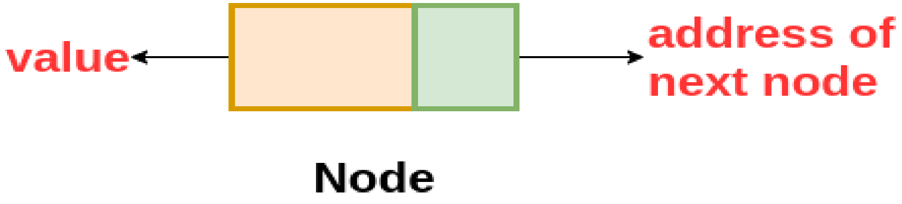
Head: pointer to the first node

The last node points to None

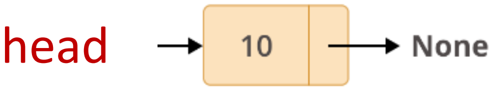


Terminology of LL

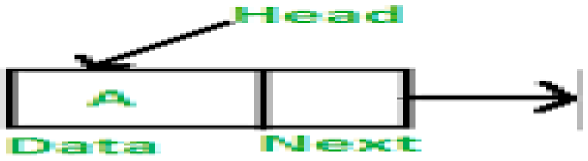
1. Node



2. None Pointer



The last node must have its next reference pointing to None to determine the end of the list.



3. External Pointer

It is a pointer to the very first node in the linked list, it enables access the entire linked list.

4.Empty List

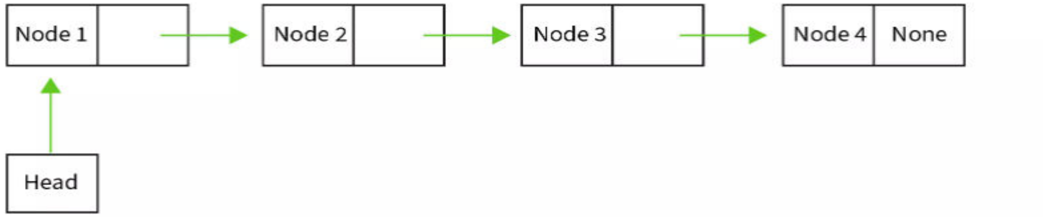
If the node are not present in the list , then it is called empty linked list.

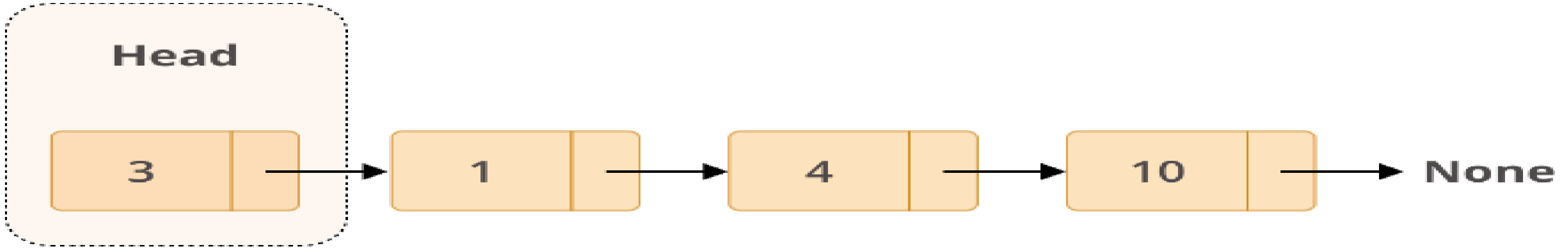
Types of LL

1. Singly Linked List. –In python ,Navigation is forward and backward manner.
2. Doubly Linked List. -Forward & Backward navigation is possible.
3. Circular Linked List. -Last element is linked to the first element.
4. Circular Doubly Linked List

Singly Linked List

- Linear singly-linked list (or simply linear list)
- A Singly-linked list is a collection of nodes linked together in a sequential way where each node of the singly linked list contains a data field and an address field that contains the reference of the next node.
- The last node is None i.e. ***next = None***, which indicates the end of the linked list.

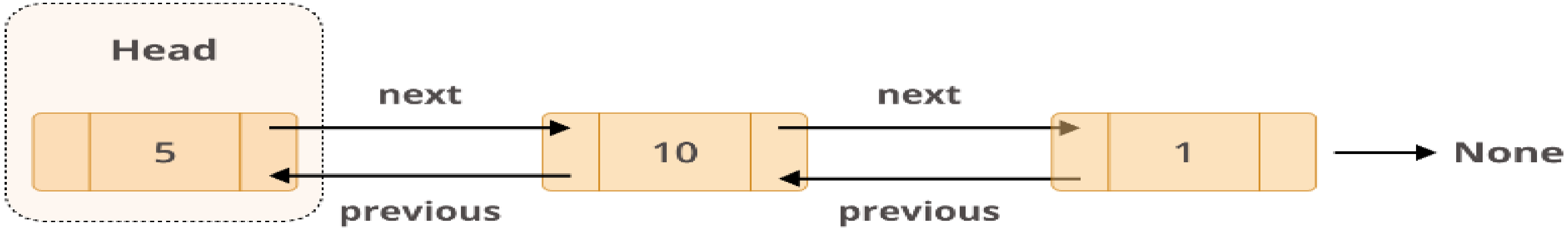




Linked List

Doubly Linked List

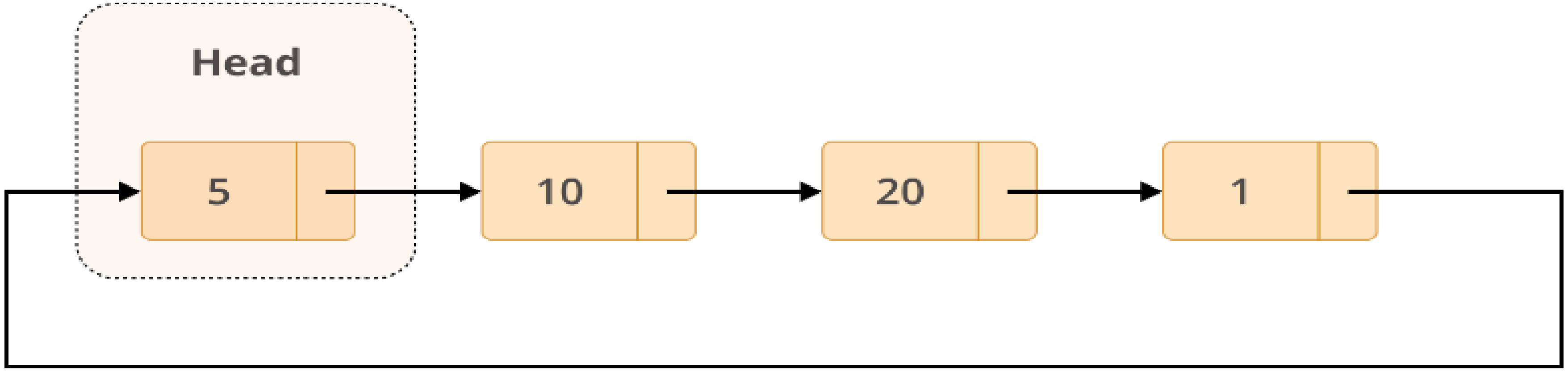
- A **doubly linked list** is a list that has two references, one to the next node and another to previous node.
- A **D**oubly **L**inked **L**ist (DLL) contains an extra pointer, typically called *previous pointer*, together with next pointer and data which are there in singly linked list.



Doubly Linked List

Circular Linked List

- ***Circular linked list*** *is a linked list where all nodes are connected to form a circle. There is no None at the end.*
- *A circular linked list can be a singly circular linked list or doubly circular linked list.*

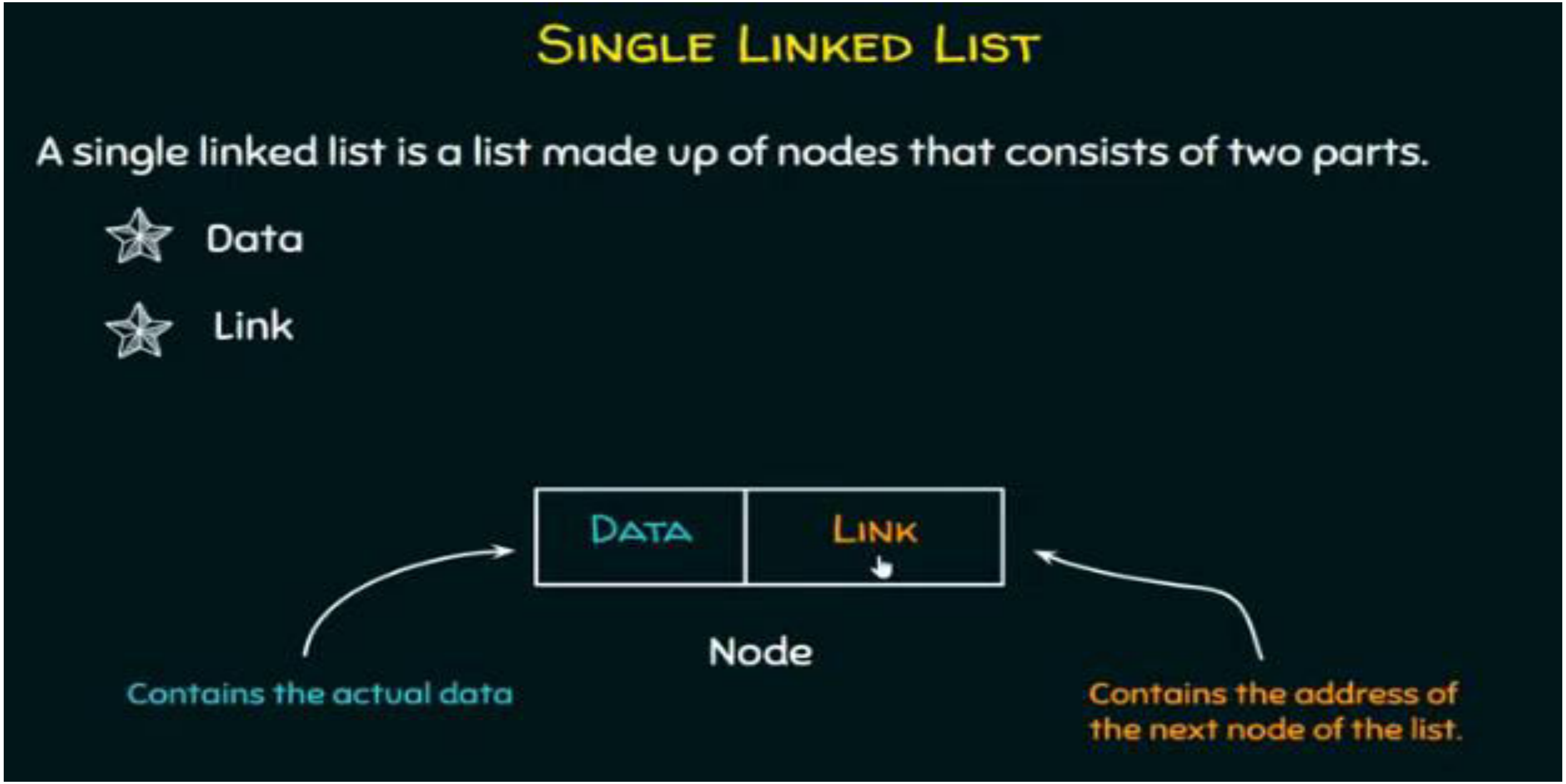


Circular Linked List

Operation of LL

1. **Traversal**: To traverse all the nodes one after another.
2. **Insertion**:
    1. To add a node at the given position.
    2. To add a node at the first position
    3. To add a node at the last position
3. **Deletion**:
    1. To Delete a node at the given position.
    2. To Delete a node at the first position
    3. To Delete a node at the last position
4. **Searching**: To search an element(s) by value.
5. **Updating**: To update a node.
6. **Sorting:** To arrange nodes in a linked list in a specific order.
7. **Merging:** To merge two linked lists into one.

Implementation of SLL



SINGLE LINKED LIST

A single linked list is a list made up of nodes that consists of two parts.

- Data
- Link

DATA | LINK

Node

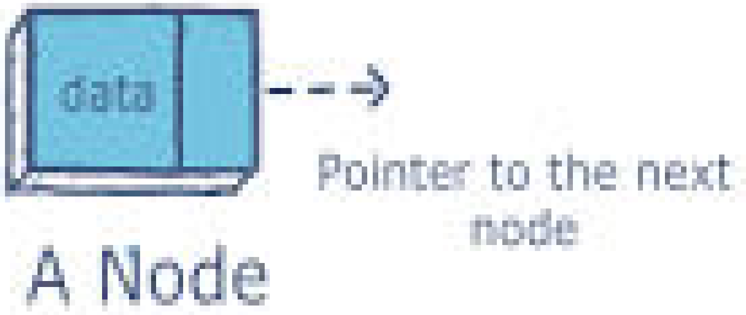
Contains the actual data

Contains the address of the next node of the list.

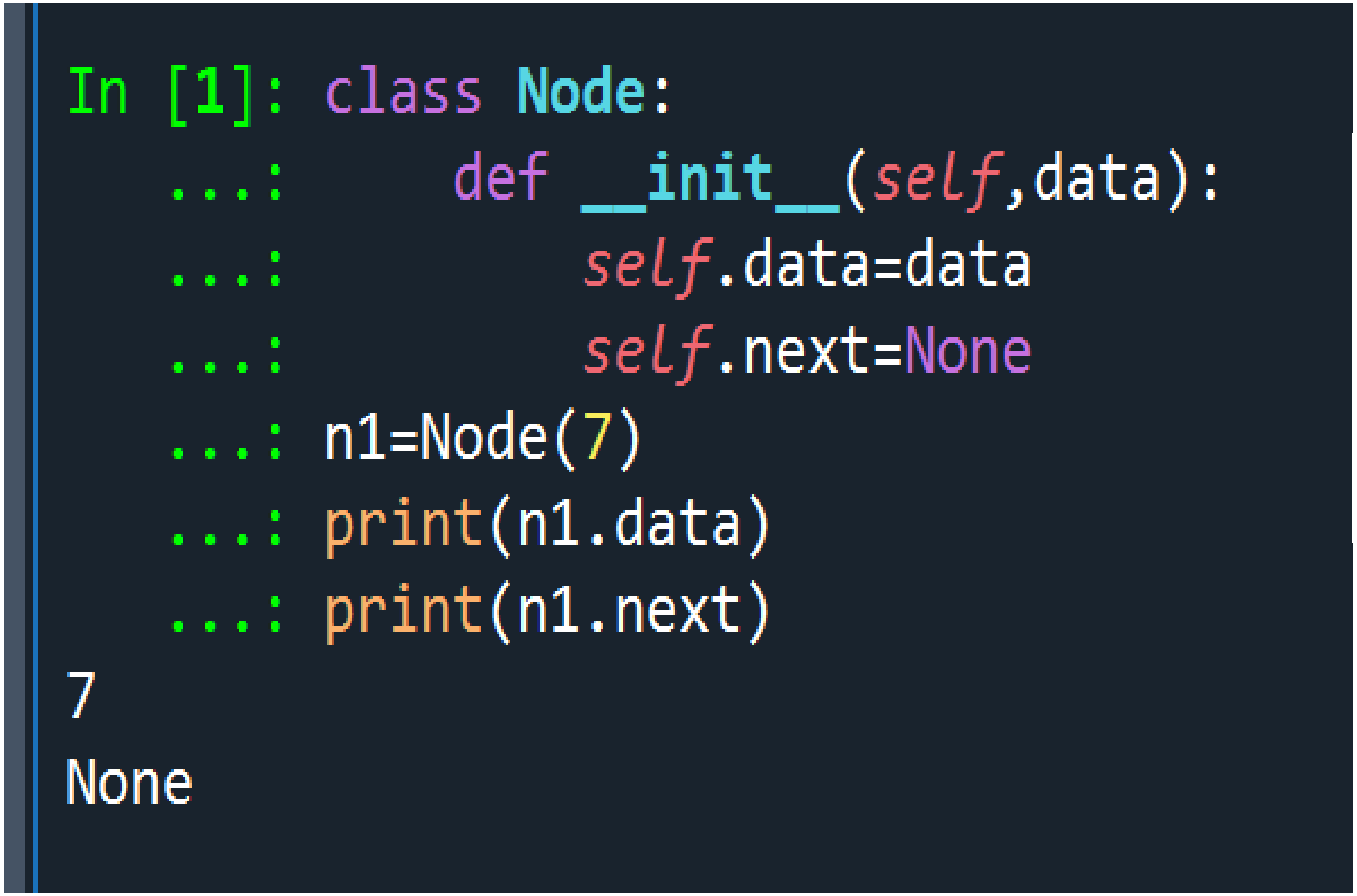
Representation of Node

- **Representation:**
  A linked list is represented by a pointer to the first node of the linked list. The first node is called the head. If the linked list is empty, then the value of the head is None.
  Each node in a list consists of at least two parts:
  1) data
  2) Pointer (Or Reference) to the next node

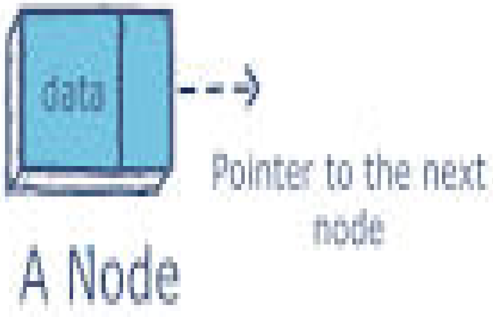
```
• // A linked list node
• Class Node
•     def __init__(self,data):
•         self.data=data;
•         self.next=None;
```



Creation of Node

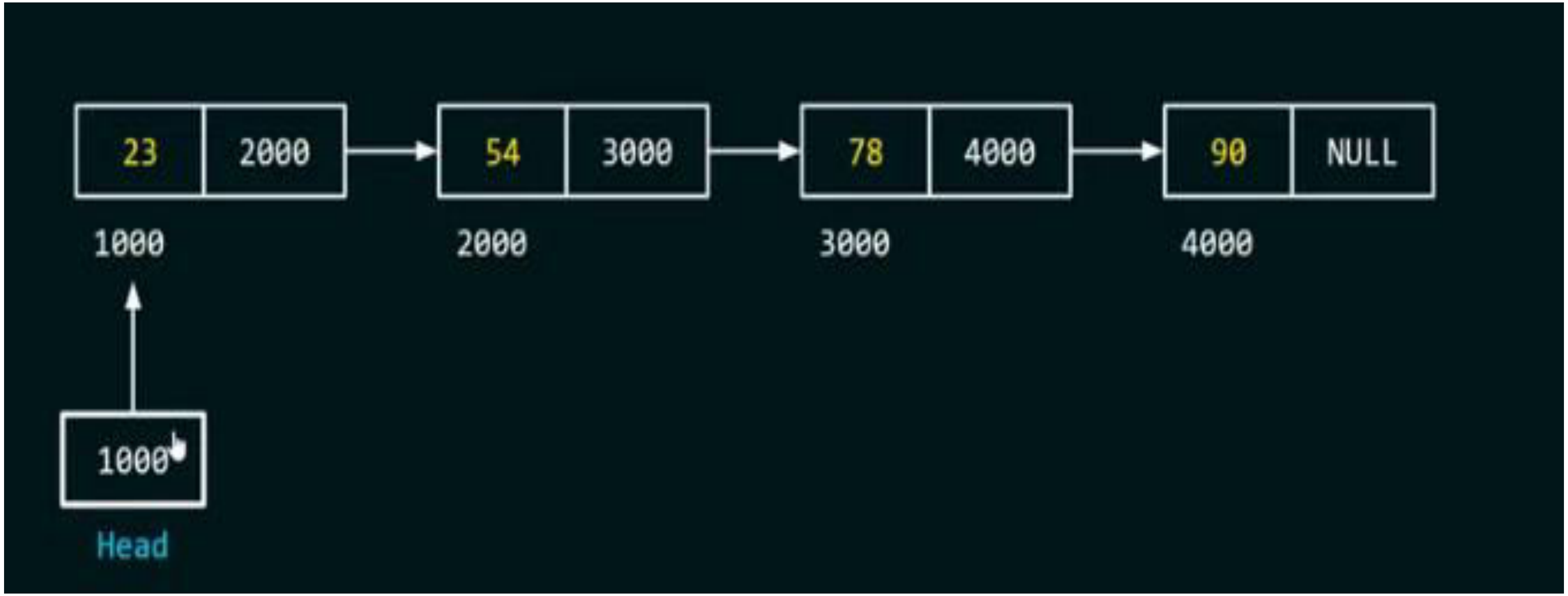


```
In [1]: class Node:
   ...:     def __init__(self,data):
   ...:         self.data=data
   ...:         self.next=None
   ...: n1=Node(7)
   ...: print(n1.data)
   ...: print(n1.next)
7
None
```



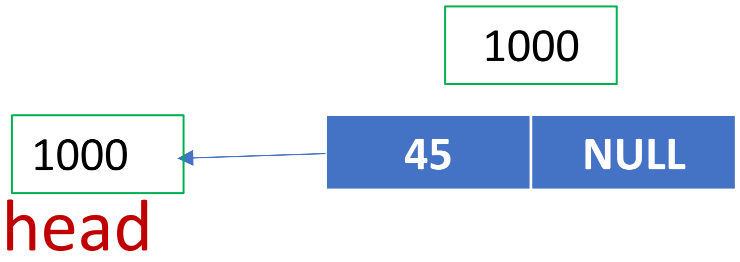
Representation of SLL

- Suppose, we want to store a list of numbers : 23,54 ,78,90



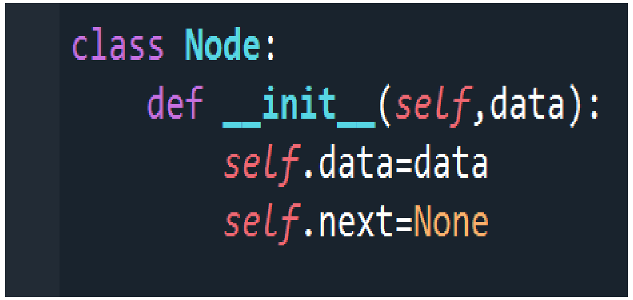
Node representing in python.

```
class Node:
    def __init__(self,data):
        self.data=data
        self.next=None
#creation of singly linked list
class SLL:
    def __init__(self):
        self.head=None
```

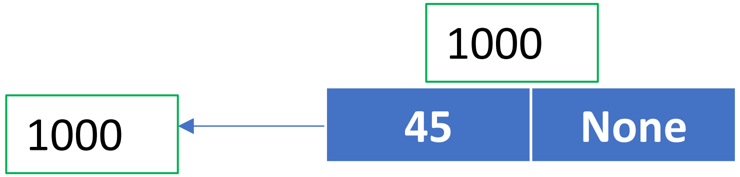


Insert at First

```
def addF(self, data):  #Algorithm for insert at Begining in SLL.
    nd = Node(data)   #Allocating new Node
    if self.head==None:   #check empty
        self.head = nd
    else: # attach beginning at head node
        nd.next = self.head
        self.head=nd
```

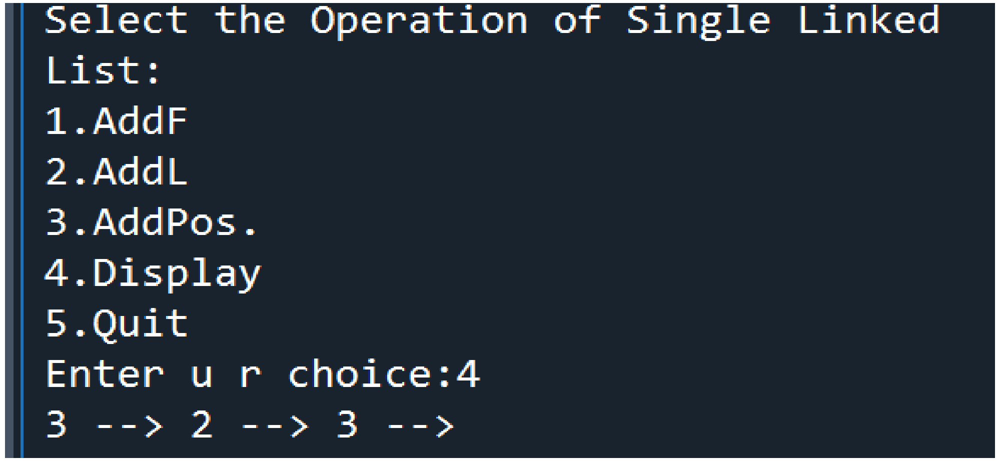


```
class Node:
    def __init__(self,data):
        self.data=data
        self.next=None
```



Display

```
def traverse(self):
    if self.head is None: [Check list is empty or not]
        print ("empty LL")
    else:   [Traverse from head to last node]
        a=self.head
        while a is not None:
            print(a.data, “--> "end=" ")
            a=a.next
```



```
Select the Operation of Single Linked
List:
1.AddF
2.AddL
3.AddPos.
4.Display
5.Quit
Enter u r choice:4
3 --> 2 --> 3 -->
```

Advantages of LL

1. They are a dynamic in nature which allocates the memory when required.
2. Insertion and deletion operations can be easily implemented.
3. Stacks and queues can be easily executed.
4. Linked List reduces the access time.

Disadvantages of LL

1. The memory is wasted as pointers require extra memory for storage.
2. No element can be accessed randomly; it has to access each node sequentially.
3. Reverse Traversing is difficult in linked list.

- Applications of Linked Lists

1. Linked lists are used to implement stacks, queues, graphs, etc.
2. Linked lists let you insert elements at the beginning and end of the list.
3. In Linked Lists we don't need to know the size in advance.

Any Questions???

Data Structure with Python
DLL

Mr. V. M. Vasava

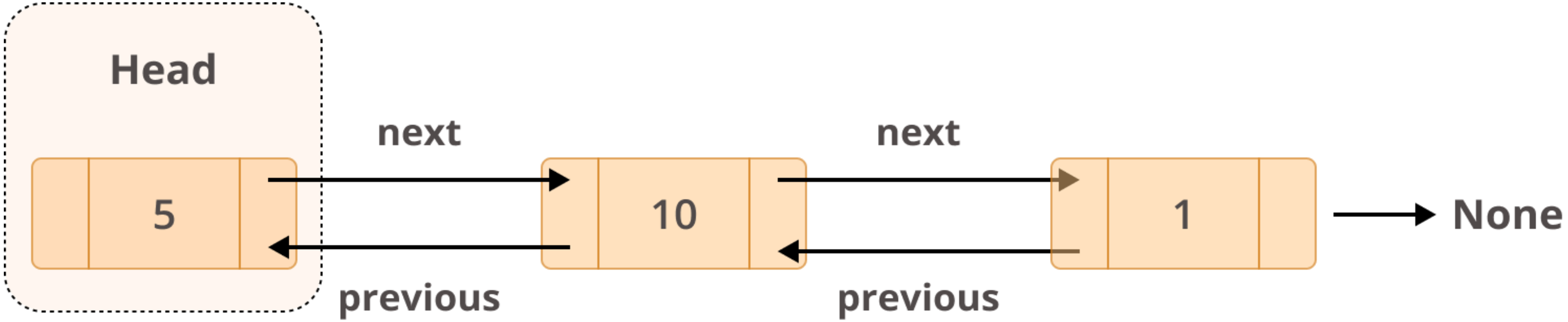
GPG,IT Dept. Surat

Agenda

- Introduction about DLL
- Representation of DLL node
- Insertion at First
- Insertion at Last

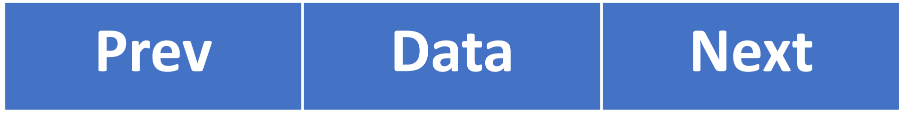
Introduction

- Doubly linked list is a type of linked list in which each node apart from storing its data has two links.
- The first link points to the previous node in the list and the second link points to the **next node** in the list.
- The first node of the list has its previous link pointing to None similarly the last node of the list has its next node pointing to None.



Implementation of node in DLL

```
class Node:
    def __init__(self,data):
        self.data=data
        self.next=None
        self.prev=None
#creation of doubly linked list
class SLL:
    def __init__(self):
        self.head=None
```

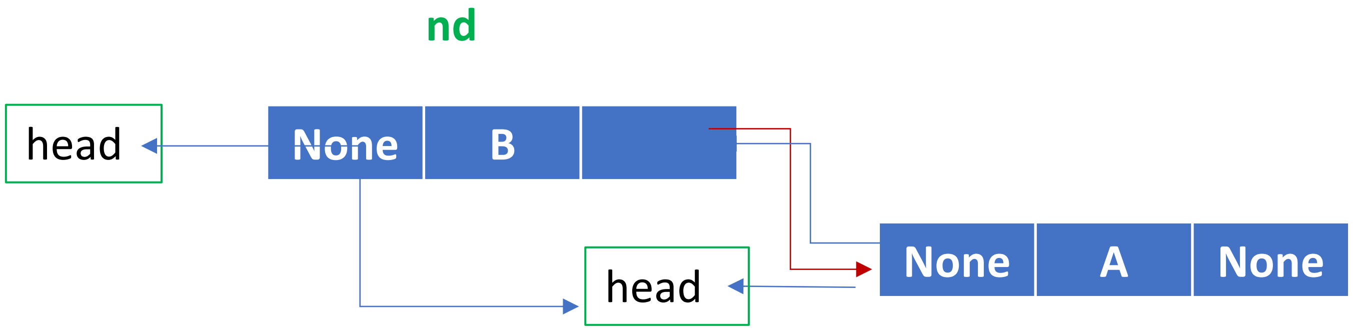


Prev	Data	Next

Insertion at First

The new node is always added at the beginning of the given Linked List.

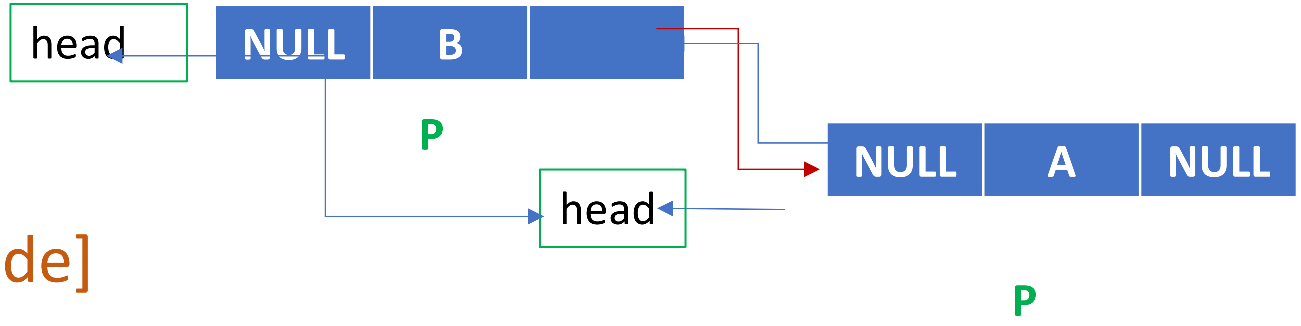
```
def addF(self, data):  #Algorithm for insert at Begining in DLL
    nd = Node(data)   #Allocating new Node
    if self.head==None:   #check empty
        self.head = nd                    P
    else: # attach beginning at head node
        nd.next = self.head
        nd.prev=None
        self.head=nd
```



Insertion at Last

The new node is always added after the last node of the given doubly Linked List.

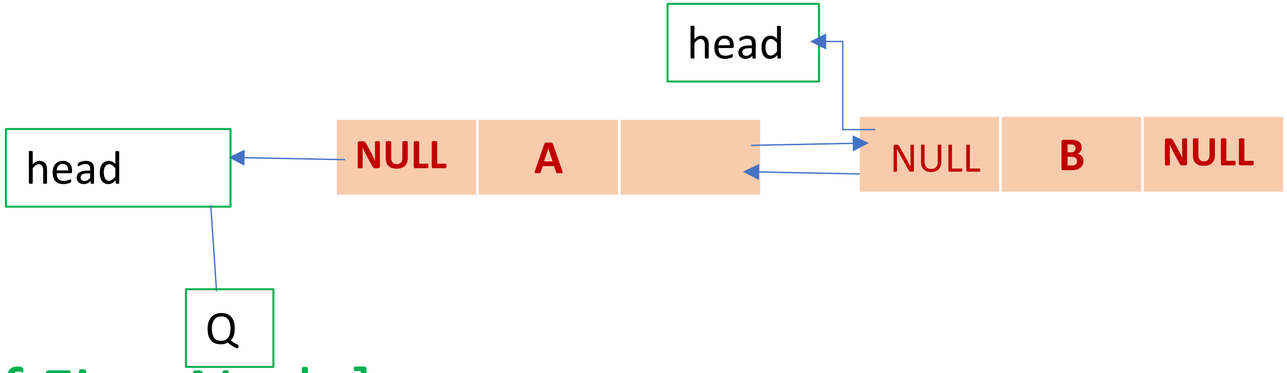
```
def addL(self,data):                    #Algorithm for insert at Last in SLL
1.Allocate Node in Memory
  nd = Node(data)   #Allocating Node
2. [Empty List]
    if self.head==None:
      self.head = nd
    else: [one or more Node]
3. [Search Last Node]
      temp=self.head
      while temp.next!=None:
        temp =temp.next
      temp.next = nd
      nd.prev=temp
```



Delete at First

Objective: Delete a node from beginning of the Doubly linked list...

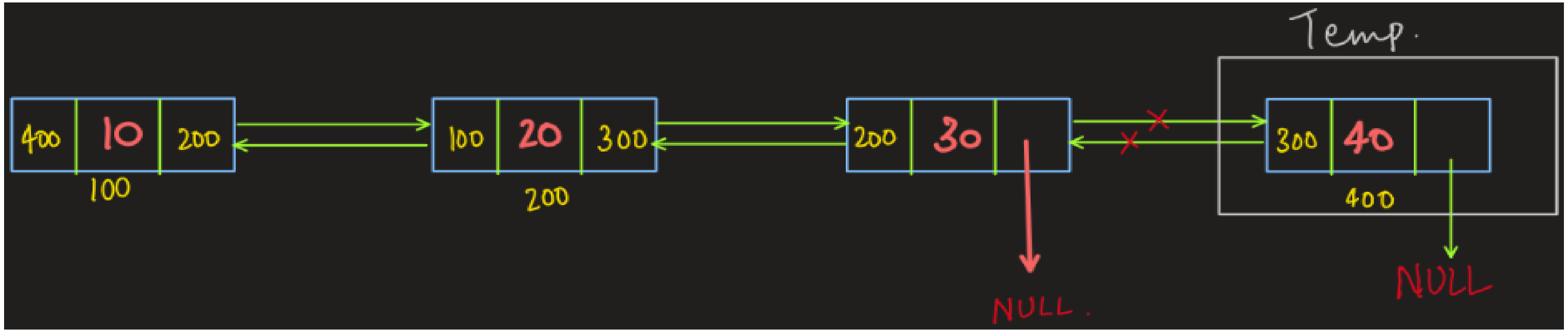
```
1.[Check The List is Empty]
     if self.head==None:
        print("List is empty")
2.    else:
        [Assign the Address of First Node]
         tmp=self.head
         print("the deleted element is",tmp.data)
         self.head=self.head.next [Move pointer from first node to next node]
        self.head.prev=None
3. Return
```



Delete at Last

Objective: Delete a node at Last of the given Doubly linked list...

- ➢We traverse through the linked list to reach the last node.
- ➢We store our last node into a temporary node.
- ➢We change the next pointer of the new **last node** to point to NULL.
- ➢We delete the temporary node.
- ➢The following diagram may help in understanding the above approach



Delete Last Node

```
def delL(self):
    if self.head is None:
        print("The list has no element to delete")
        return
    if self.head.next is None:
        self.head = None
        return
    n = self.head
    while n.next is not None:
        n = n.next
    n.prev.next = None
```

Advantages

- Allows traversal of nodes in both direction which is not possible in singly linked list.
- Deletion of nodes is easy when compared to singly linked list, as in singly linked list deletion requires a pointer to the node and previous node to be deleted. Which is not in case of doubly linked list we only need the pointer which is to be deleted.
- Reversing the list is simple and straightforward.
- Can allocate or de-allocate memory easily when required during its execution.

Disadvantages

1. It requires more space per space per node because one extra field is required for pointer to previous node.

2. Insertion and deletion take more time than linear linked list because more pointer operations are required than linear linked list.

3.Since elements in memory are stored randomly, hence elements are accessed sequentially no direct access is allowed.

Any Questions??????

Data Structure

Click to add text

Linked List

Mr. V. M. Vasava

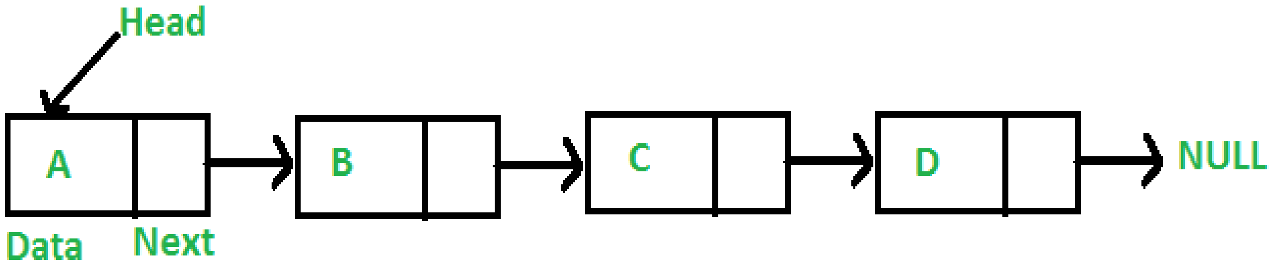
GPG,DIT,Surat.

Agenda

- Introduction
- Types of Linked List
- Operation of Linked List

Linked List

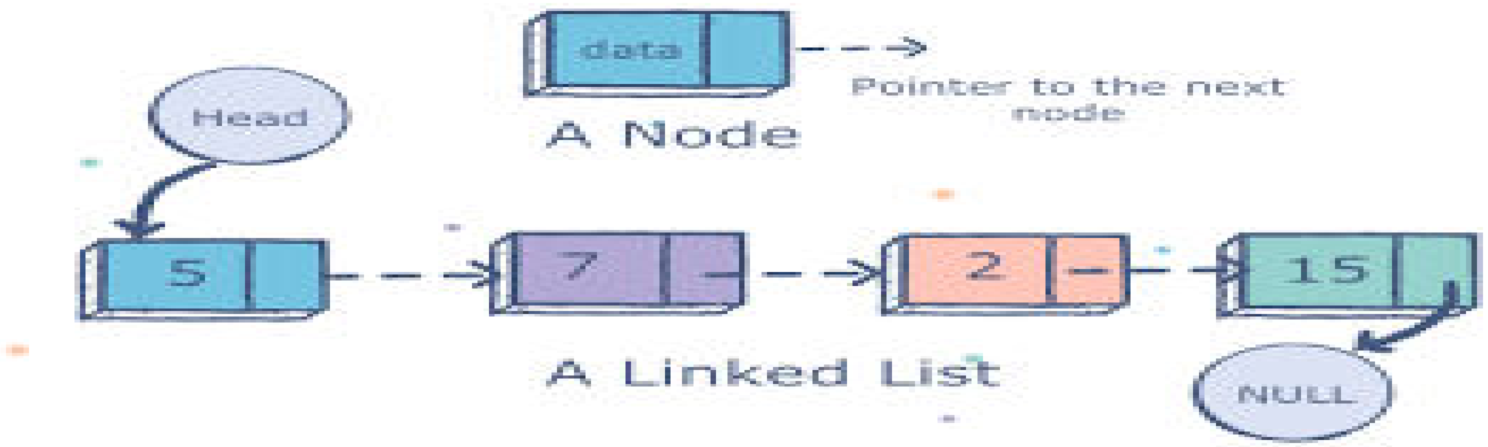
- A linked list is a linear data structure, in which the elements are stored at randomly in memory locations.it consists of nodes where each node contains a data field and a reference which contains the address of next node. The elements in a linked list are linked using pointers as shown in the below image:



- a linked list consists of nodes where each node contains a data field and a reference(link) to the next node in the list.
- A linked list is a sequence of data structures, which are connected together via links.

Linked List

- It is collection or group of nodes.
- Each node of a **list** is made up of two items - the data and a reference which contains address of next nodes.

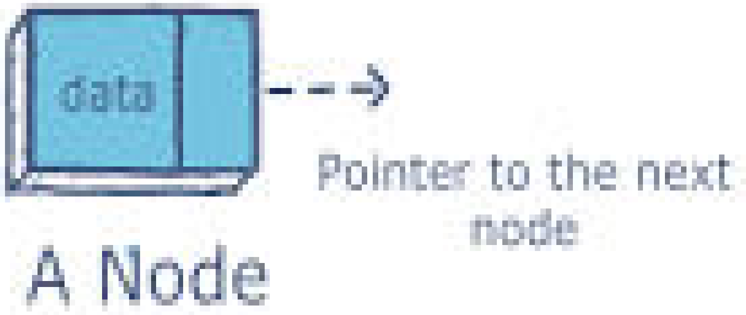


- It is a linear data structure.
- **Linked list** elements are stored at randomly in memory.

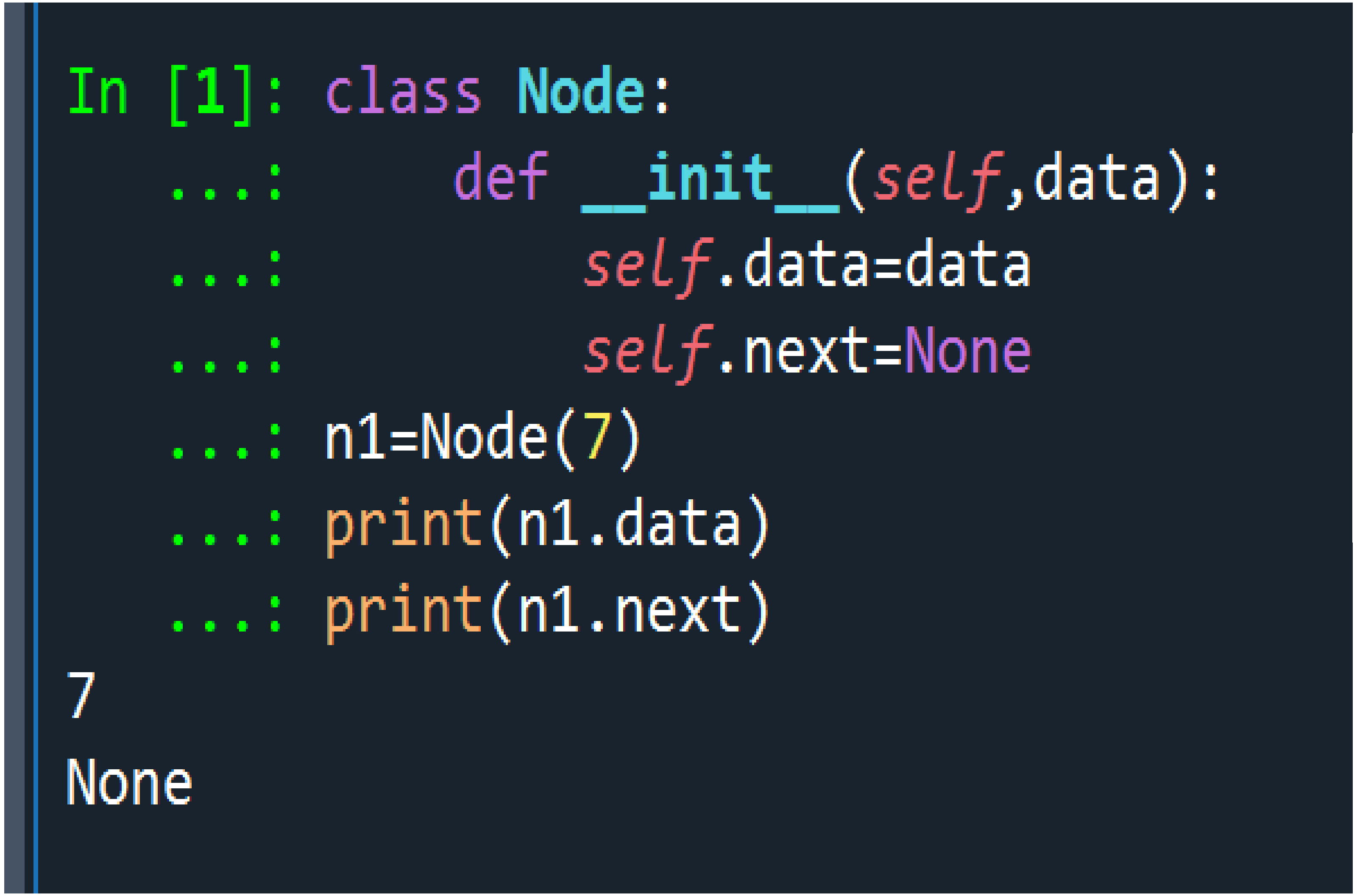
Representation of Node

- **Representation:**
  A linked list is represented by a pointer to the first node of the linked list. The first node is called the head. If the linked list is empty, then the value of the head is None.
  Each node in a list consists of at least two parts:
  1) data
  2) Pointer (Or Reference) to the next node

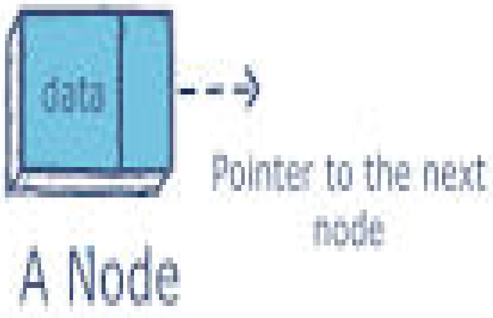
```
// A linked list node
Class Node
    def __init__(self,data):
        self.data=data;
        self.next=None;
```



Creation of Node



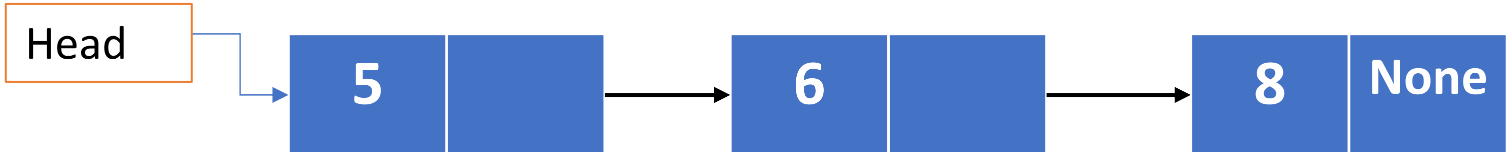
```
In [1]: class Node:
   ...:     def __init__(self,data):
   ...:         self.data=data
   ...:         self.next=None
   ...: n1=Node(7)
   ...: print(n1.data)
   ...: print(n1.next)
7
None
```



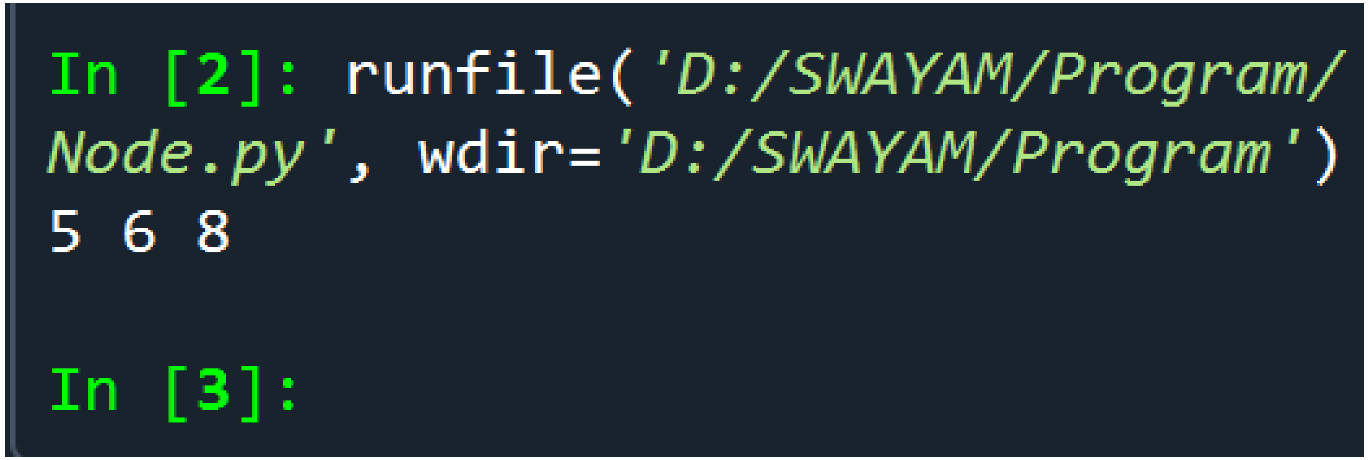
Creation of Node

```
class Node:
    def __init__(self,data):
        self.data=data
        self.next=None
#creating linked list
class SLL:
    def __init__(self):
        self.head=None
```

```
def traverse(self):
    if self.head is None:
      print ("empty")
    else:
     a =self.head
     while a is not None:
      print(a.data,end=" ")
      a=a.next
```



```
sll=SLL()
n1=Node(5)
sll.head=n1
n2=Node(6)
n1.next=n2
n3=Node(8)
n2.next=n3
sll.traverse()
```



```
In [2]: runfile('D:/SWAYAM/Program/
Node.py', wdir='D:/SWAYAM/Program')
5 6 8

In [3]:
```

Advantages of LL

1. They are a dynamic in nature which allocates the memory when required.
2. Insertion and deletion operations can be easily implemented.
3. Stacks and queues can be easily executed.
4. Linked List reduces the access time.

Disadvantages of LL

1. The memory is wasted as pointers require extra memory for storage.
2. No element can be accessed randomly; it has to access each node sequentially.
3. Reverse Traversing is difficult in linked list.

- Applications of Linked Lists

1. Linked lists are used to implement stacks, queues, graphs, etc.
2. Linked lists let you insert elements at the beginning and end of the list.
3. In Linked Lists we don't need to know the size in advance.

Any Questions????

Data Structure with Python
SLL

Mr. V. M. Vasava

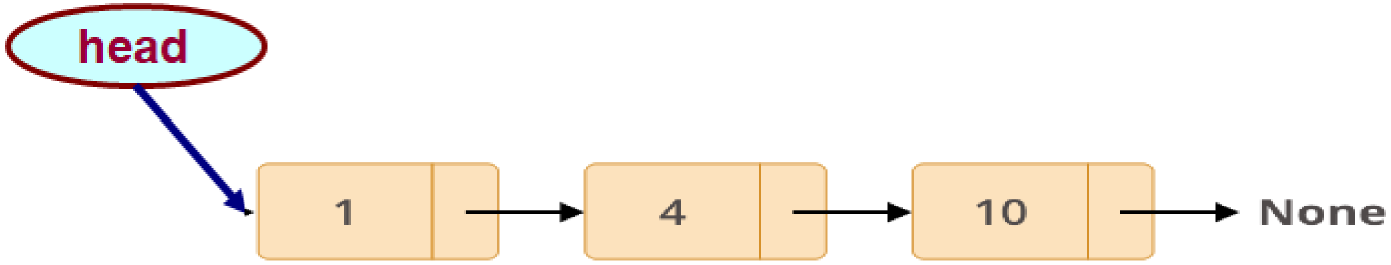
GPG,IT Dept. Surat

Agenda

- Introduction about SLL
- Insertion at Last
- Insertion at Position
- Traversing
- Deletion at First

SLL(single linked list)

- singly linked list is a linear data structure in which each element of the list contains a pointer which points to the next element in the list.
- –Successive elements are connected by pointers.
- –Last element points to None.
- –It can grow or shrink in size during execution of a program.
- –It can be made just as long as required.
- –It does not waste memory space.



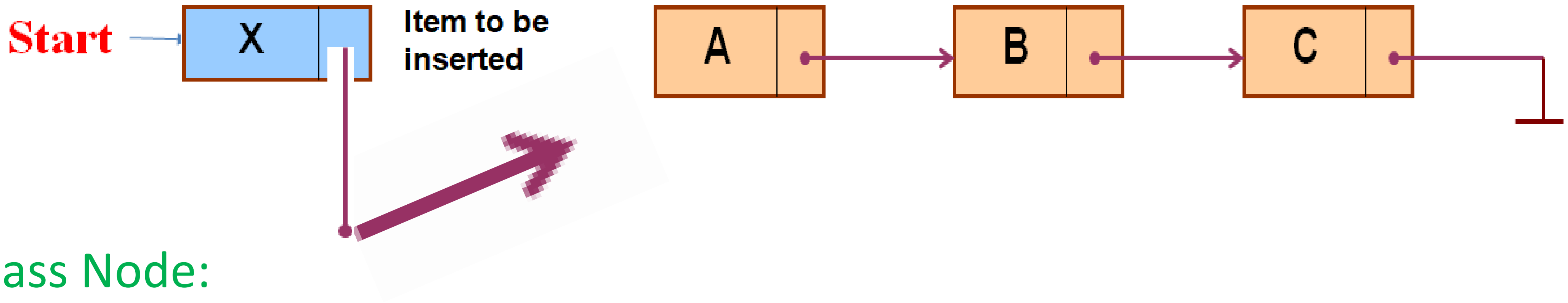
Linked List

Continued..

- ✓**Single linked list is a sequence of elements in which every element has link to its next element in the sequence.**
- ✓Keeping track of a linked list:
- ✓–Must know the pointer to the first element of the list (called *start*, *head*, etc.).
- ✓Linked lists provide flexibility in allowing the items to be rearranged efficiently.
- ✓–Insert an element.
- ✓–Delete an element.

illustration: Insertion at First

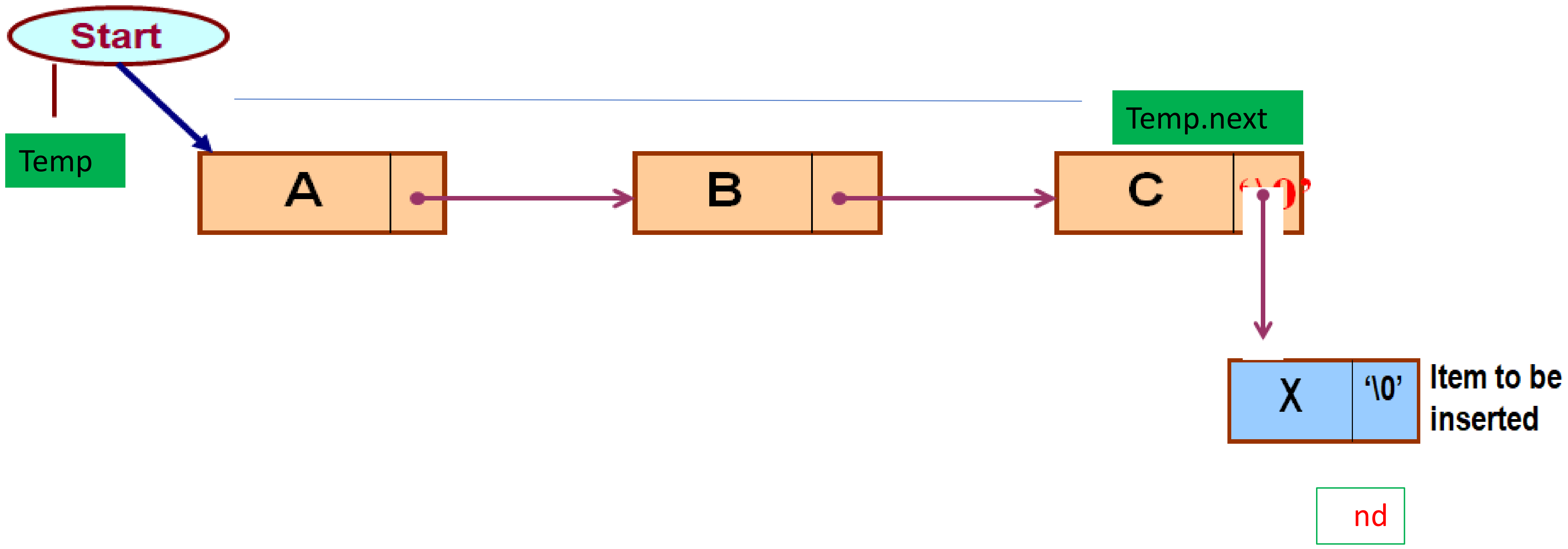
- Insertion at First



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

illustration:Insertion at Last

- Insertion at Last



Algorithm:InsertionLast()

- The new node is always added after the last node of the given Linked List.

```
def addL(self,data):                 #Algorithm for insert at Last in SLL
1.Allocate Node in Memory
   nd = Node(data)   #Allocating Node
2. [Empty List]
      if self.head==None:
         self.head = nd
      else: [one or more Node]
3. [Search Last Node]
         temp=self.head
         while temp.next!=None:
            temp =temp.next
         temp.next = nd
```

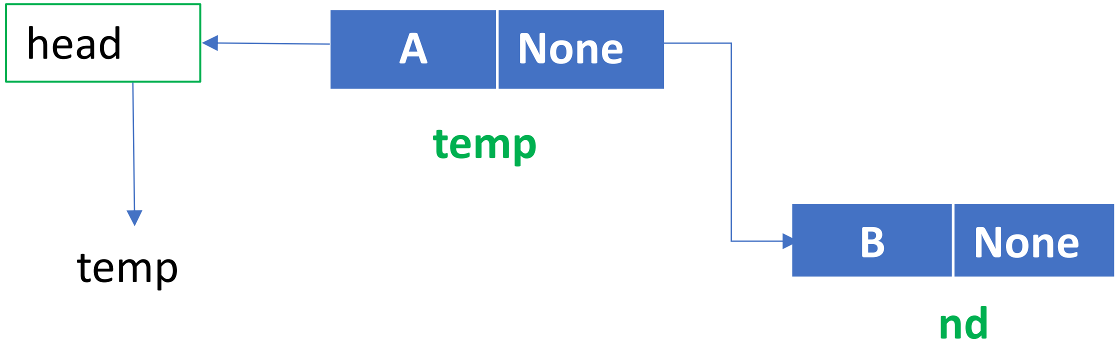
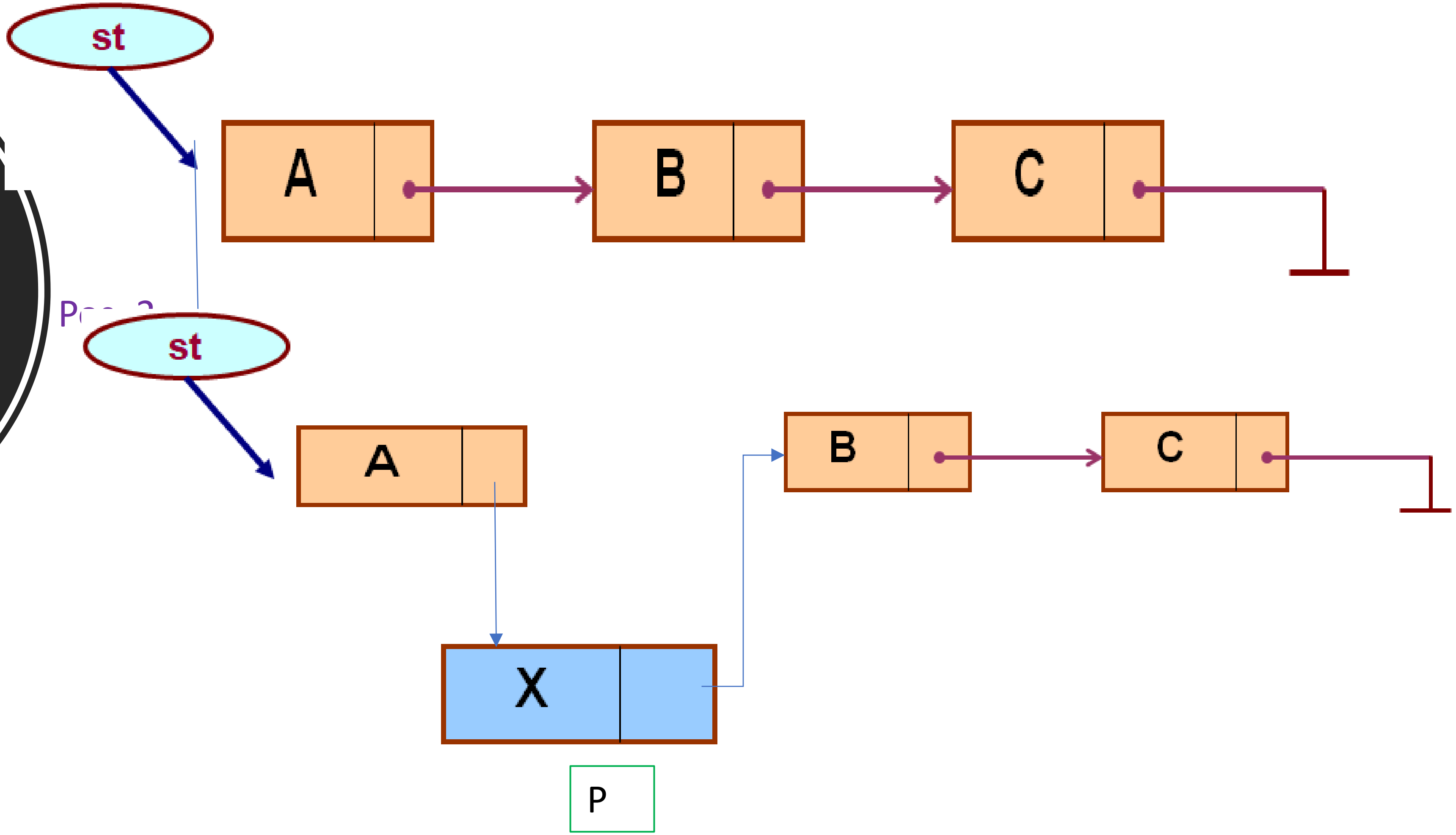


Illustration :Insertion at Position

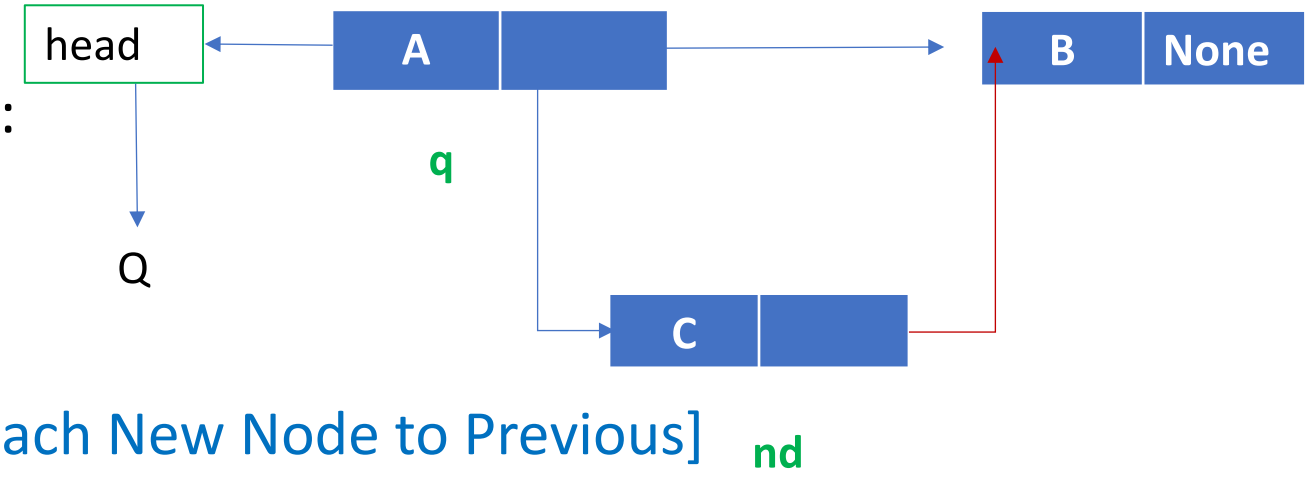
- Insertion at Specified Position



Algorithm:InsertionPosition(data,pos)

- The new node is always added at the specified at given position inLinked List.

```
def addP(self,data,pos):#Algorithm for insert at Last in SLL
 1.      nd = Node(data)   #Allocating Node
 2.      if pos==1: .[Position =1(First)]
           nd.next=self.head
           self.head=nd
3. else:[Search  Node at Given Position ]
         q=self.head
         for i in range(1,pos-1):
            if (q!=None):
               q=q.next
         if (q!=None):
            nd.next=q.next [Attach New Node to Previous]
            q.next=nd [Attach New Node to Next]
         else:
            print("previous node is null")
```



Traverse SLL

Algo: Display() Print the data of all the elements

Print the number of the nodes in the list

```
def traverse(self):
1.   if self.head is None: [Check list is empty or not]
         print ("empty LL")
2.     else:   [Traverse from head to last node]
         a=self.head
         while a is not None:
             print(a.data,end=" ")
             a=a.next
```

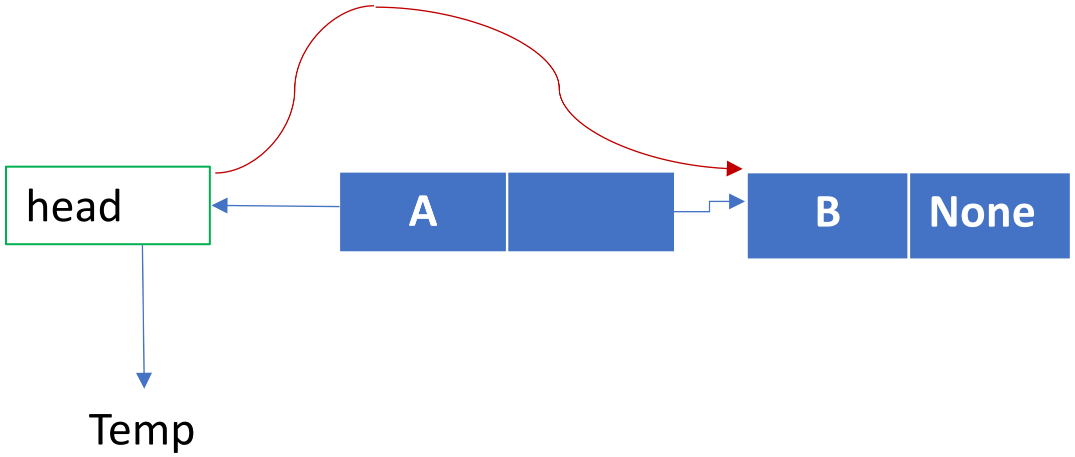
Assignments

- 1. Write an algorithm for Count the Total number of nodes in List.
- 2.Find the middle of a given linked list in python.
- 3.Write a function that counts the number of times a given int occurs in a Linked List.

Algorithm:DeleteFirst()

- Objective: Delete a node from beginning of the single linked list...

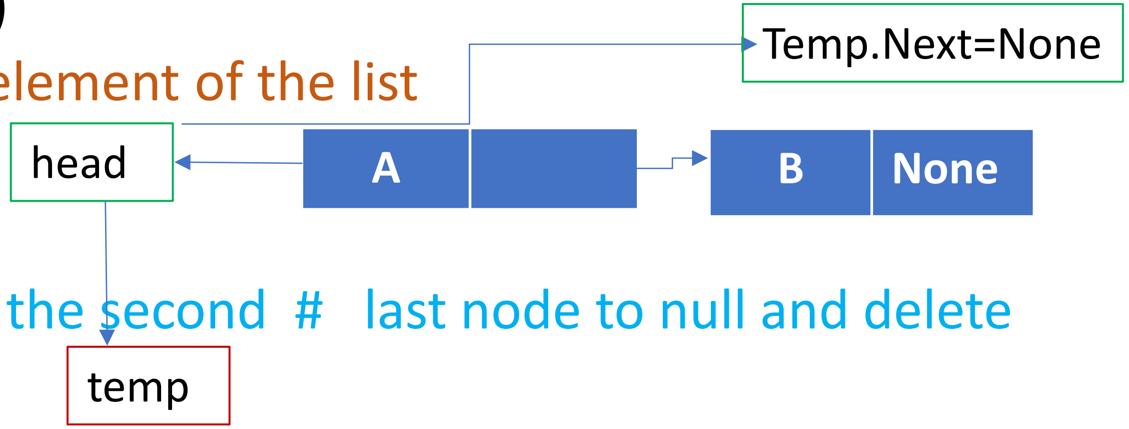
```
 def delF(self):
1.[Check The List is Empty]
      if self.head==None:
         print("List is empty")
2.    else:
         [Assign the Address of First Node]
         tmp=self.head
         print("the deleted element is",tmp.data)
         self.head=self.head.next [Move pointer from first node to next node]
3. Return
```



Algorithm:DeleteLast()

- Objective: Delete a node from Last of the single linked list...

```
def delL(self):
1. if(self.head != None):       #1. if head in not null and next of head    #   is null, release the head
        if(self.head.next == None):
            self.head = None
            print("The deleted node is ->",self.head.data)
 2.     else:   #2. Else, traverse to the second last   #   element of the list
            temp = self.head
            while(temp.next != None):
                temp = temp.next   #3. Change the next of the second  #   last node to null and delete
the last node
            lastNode = temp.next
            print("The deleted node last is:-",lastNode.data)
            temp.next = None
            lastNode = None
     else:
        print("List is empty")
```



Array Vs Linked List

Arrays are suitable for:

–Inserting/deleting an element at the end.

–Randomly accessing any element.

–Searching the list for a particular value.

Linked lists are suitable for:

–Inserting an element.

–Deleting an element.

–Applications where sequential access is required.

–In situations where the number of elements cannot be predicted beforehand.

Advantages of SLL

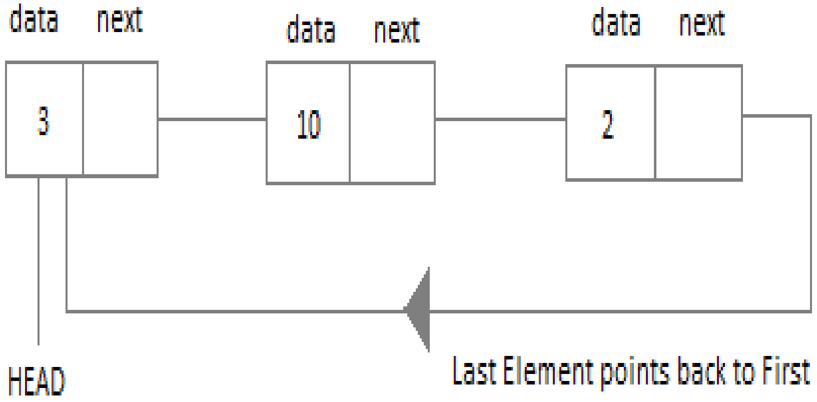
1) Insertions and Deletions can be done easily.
2) It does not need movement of elements for insertion and deletion.
3) It space is not wasted as we can get space according to our requirements.
4) Its size is not fixed.
5) It can be extended or reduced according to requirements.
6) Elements may or may not be stored in consecutive memory available,even then we can store the data in computer.
7) It is less expensive.

Disadvantages of SLL

1) It requires more space as pointers are also stored with information.
2) Different amount of time is required to access each element.
3) If we have to go to a particular element then we have to go through all those elements that come before that element.
4) we can not traverse it from last & only from the beginning.
5) It is not easy to sort the elements stored in the linear linked list.

CLL(Circular Linked List)

- **Circular Linked List** is a variation of **Linked list** in which the first element points to the last element and the last element points to the first element.
- Both Singly **Linked List** and Doubly **Linked List** can be made into a **circular linked list**.



Operations on the circular linked list:

- circular linked list similar to the singly linked list which are:

1.Insertion

2.Deletion

3.Display

4.Sorting

Any Questions ???

Data Structure with Python
SLL

Mr. V. M. Vasava

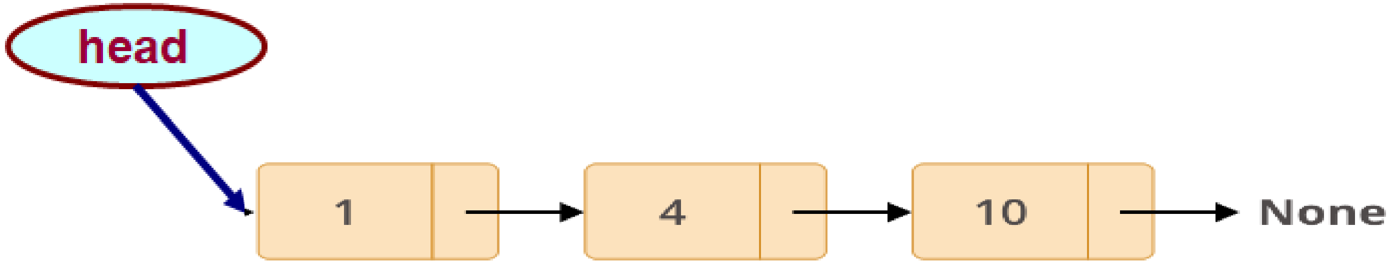
GPG,IT Dept. Surat

Agenda

- Introduction about SLL
- Insertion at Last
- Insertion at Position
- Traversing
- Deletion at First

SLL(single linked list)

- singly linked list is a linear data structure in which each element of the list contains a pointer which points to the next element in the list.
- –Successive elements are connected by pointers.
- –Last element points to None.
- –It can grow or shrink in size during execution of a program.
- –It can be made just as long as required.
- –It does not waste memory space.



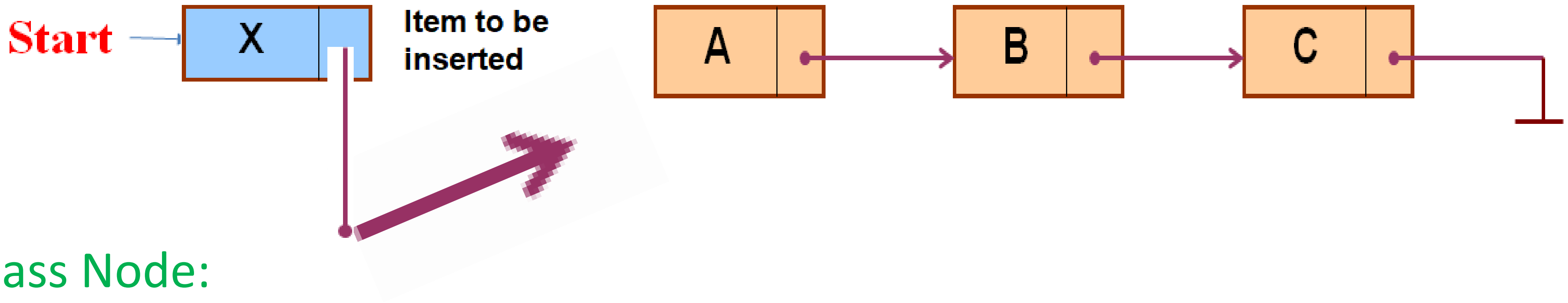
Linked List

Continued..

- ✓**Single linked list is a sequence of elements in which every element has link to its next element in the sequence.**
- ✓Keeping track of a linked list:
- ✓–Must know the pointer to the first element of the list (called *start*, *head*, etc.).
- ✓Linked lists provide flexibility in allowing the items to be rearranged efficiently.
- ✓–Insert an element.
- ✓–Delete an element.

illustration: Insertion at First

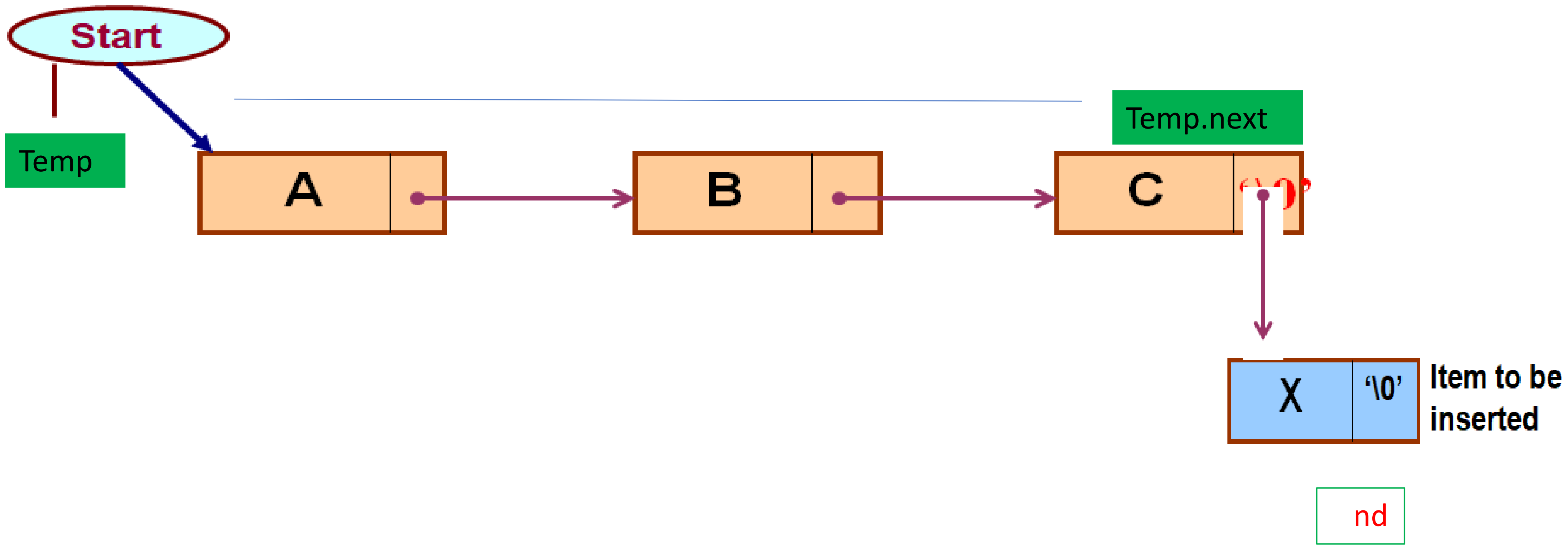
- Insertion at First



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

illustration:Insertion at Last

- Insertion at Last



Algorithm:InsertionLast()

- The new node is always added after the last node of the given Linked List.

```
def addL(self,data):                #Algorithm for insert at Last in SLL
1.Allocate Node in Memory
   nd = Node(data)   #Allocating Node
2. [Empty List]
      if self.head==None:
         self.head = nd
      else: [one or more Node]
3. [Search Last Node]
         temp=self.head
         while temp.next!=None:
            temp =temp.next
         temp.next = nd
```

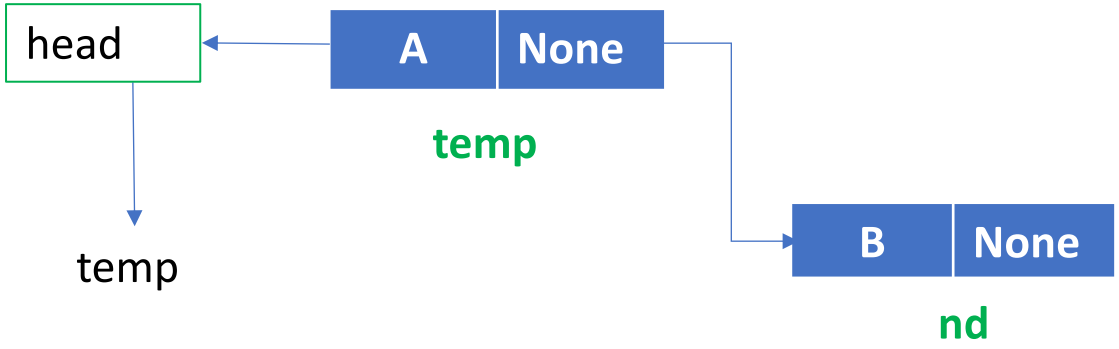
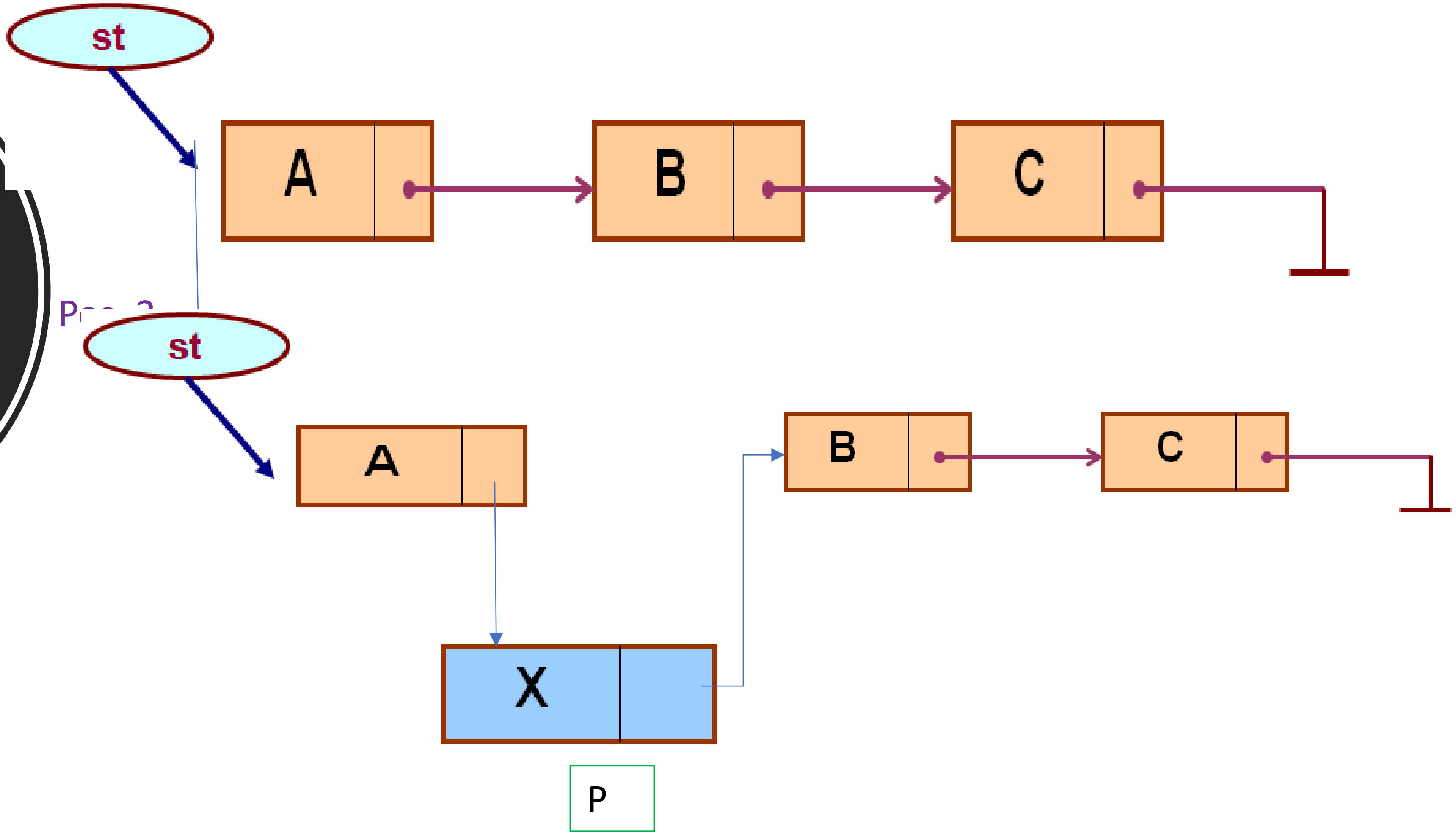


Illustration :Insertion at Position

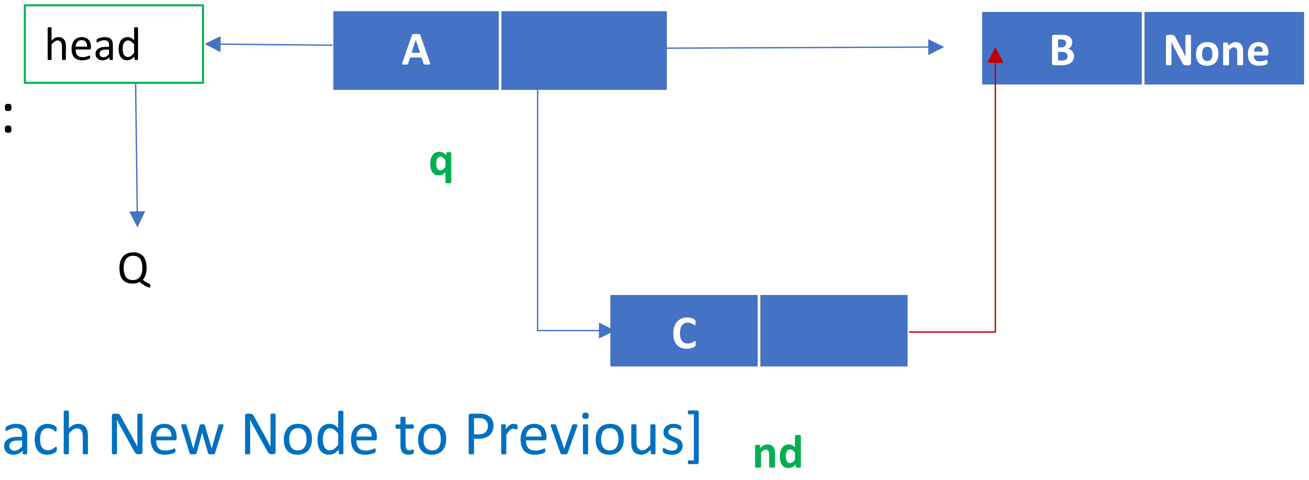
- Insertion at Specified Position



Algorithm:InsertionPosition(data,pos)

- The new node is always added at the specified at given position inLinked List.

```
def addP(self,data,pos):#Algorithm for insert at Last in SLL
 1.      nd = Node(data)   #Allocating Node
 2.      if pos==1: .[Position =1(First)]
           nd.next=self.head
           self.head=nd
3. else:[Search  Node at Given Position ]
          q=self.head
          for i in range(1,pos-1):
             if (q!=None):
                q=q.next
          if (q!=None):
             nd.next=q.next [Attach New Node to Previous]
             q.next=nd [Attach New Node to Next]
          else:
             print("previous node is null")
```



Traverse SLL

Algo: Display() Print the data of all the elements

Print the number of the nodes in the list

```
def traverse(self):
1.   if self.head is None: [Check list is empty or not]
         print ("empty LL")
2.     else:   [Traverse from head to last node]
         a=self.head
         while a is not None:
             print(a.data,end=" ")
             a=a.next
```

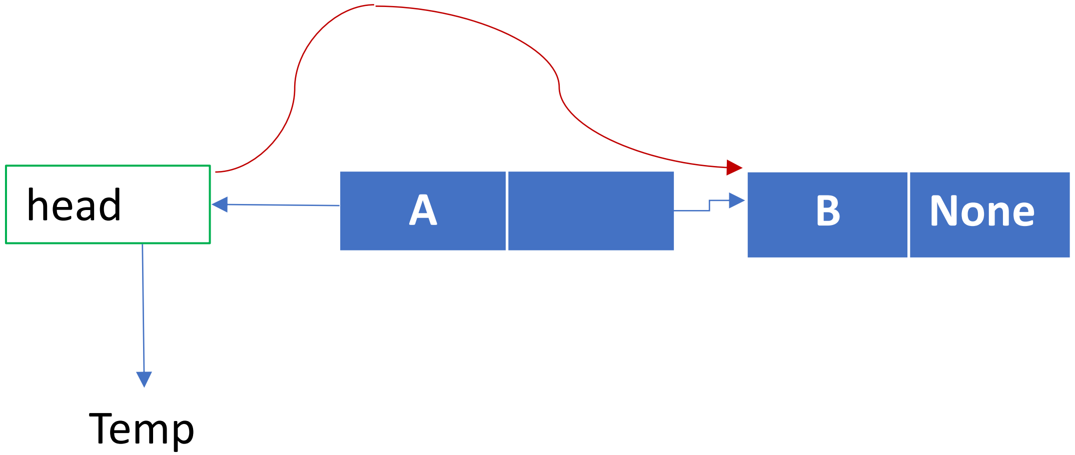
Assignments

- 1. Write an algorithm for Count the Total number of nodes in List.
- 2.Find the middle of a given linked list in python.
- 3.Write a function that counts the number of times a given int occurs in a Linked List.

Algorithm:DeleteFirst()

- Objective: Delete a node from beginning of the single linked list...

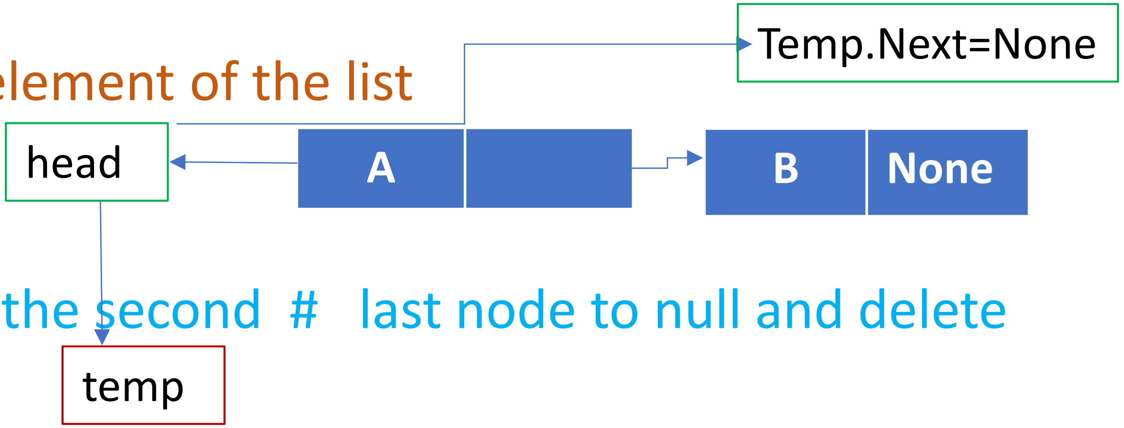
```
 def delF(self):
1.[Check The List is Empty]
      if self.head==None:
         print("List is empty")
2.    else:
         [Assign the Address of First Node]
         tmp=self.head
         print("the deleted element is",tmp.data)
         self.head=self.head.next [Move pointer from first node to next node]
3. Return
```



Algorithm:DeleteLast()

- Objective: Delete a node from Last of the single linked list...

```
def delL(self):
1. if(self.head != None):      #1. if head in not null and next of head    #   is null, release the head
        if(self.head.next == None):
          self.head = None
          print("The deleted node is ->",self.head.data)
 2.     else:   #2. Else, traverse to the second last   #   element of the list
          temp = self.head
          while(temp.next.next != None):
            temp = temp.next   #3. Change the next of the second  #   last node to null and delete the last node
          lastNode = temp.next
          print("The deleted node last is:-",lastNode.data)
          temp.next = None
          lastNode = None
    else:
        print("List is empty")
```



Array Vs Linked List

Arrays are suitable for:

–Inserting/deleting an element at the end.

–Randomly accessing any element.

–Searching the list for a particular value.

Linked lists are suitable for:

–Inserting an element.

–Deleting an element.

–Applications where sequential access is required.

–In situations where the number of elements cannot be predicted beforehand.

Advantages of SLL

1) Insertions and Deletions can be done easily.

2) It does not need movement of elements for insertion and deletion.

3) It space is not wasted as we can get space according to our requirements.

4) Its size is not fixed.

5) It can be extended or reduced according to requirements.

6) Elements may or may not be stored in consecutive memory available,even then we can store the data in computer.

7) It is less expensive.

Disadvantages of SLL

1) It requires more space as pointers are also stored with information.
2) Different amount of time is required to access each element.
3) If we have to go to a particular element then we have to go through all those elements that come before that element.
4) we can not traverse it from last & only from the beginning.
5) It is not easy to sort the elements stored in the linear linked list.

Any Questions ???