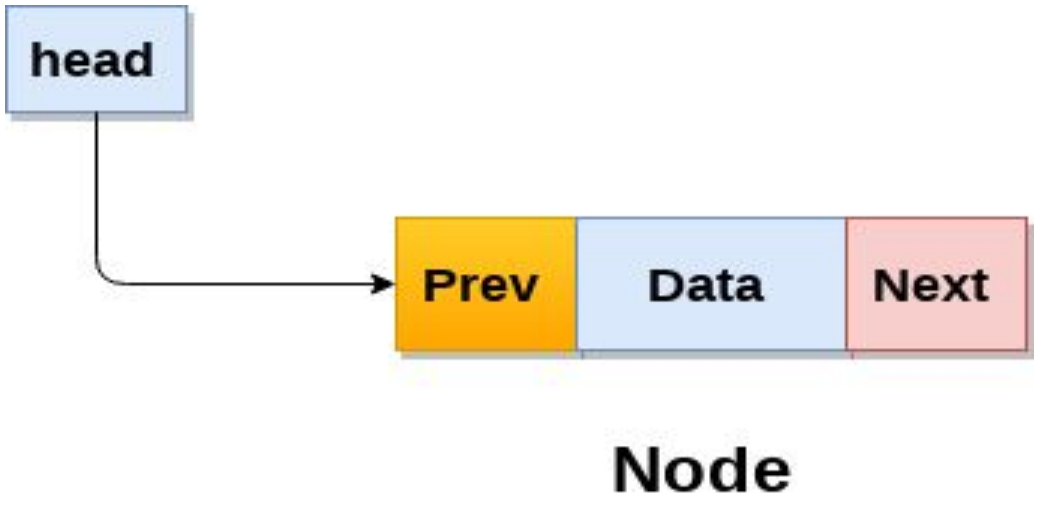
Doubly Linked List

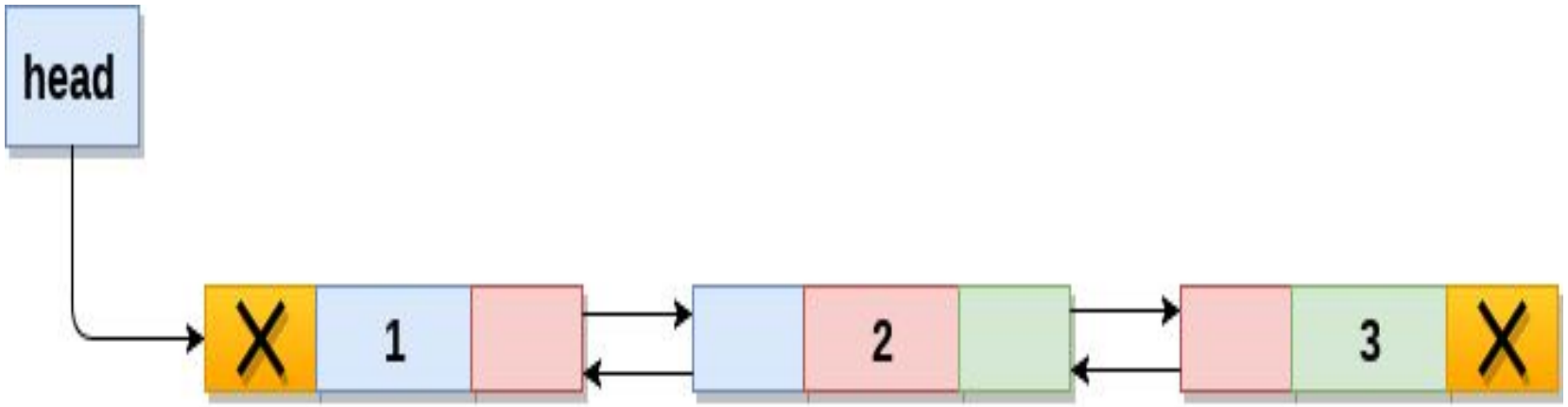
- Doubly linked list is a complex type of linked list in which a node contains a pointer to the previous as well as the next node in the sequence.
- Therefore, in a doubly linked list, a node consists of three parts: node data, pointer to the next node in sequence (next pointer) , pointer to the previous node (previous pointer).
- A sample node in a doubly linked list is shown in the figure.



Node

Doubly Linked List

A doubly linked list containing three nodes is shown in the following image.



Doubly Linked List

Creating a Node of Doubly Linked List

- Create a class for creating a node in a doubly linked list, with three attributes: the data, previous pointer and next pointer. The code looks like this:

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

Creating Doubly Linked List Class

Create a doublyLinkedList class, that contains different functions to insert, delete and display elements of doubly linked list.

```
class doublyLinkedList:
    def __init__(self):
        self.start_node = None
```

Creating a Doubly linked list with single node

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

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

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

Creation and Traversal of Doubly linked list

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

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

```
# creation method for the doubly linked list
    def create(self, data):
        newNode = Node(data)
        if(self.head==None):
            self.head = newNode

        else:
            temp=self.head
            while(temp.next!=None):
                temp=temp.next

            temp.next=newNode
            newNode.prev=temp
```

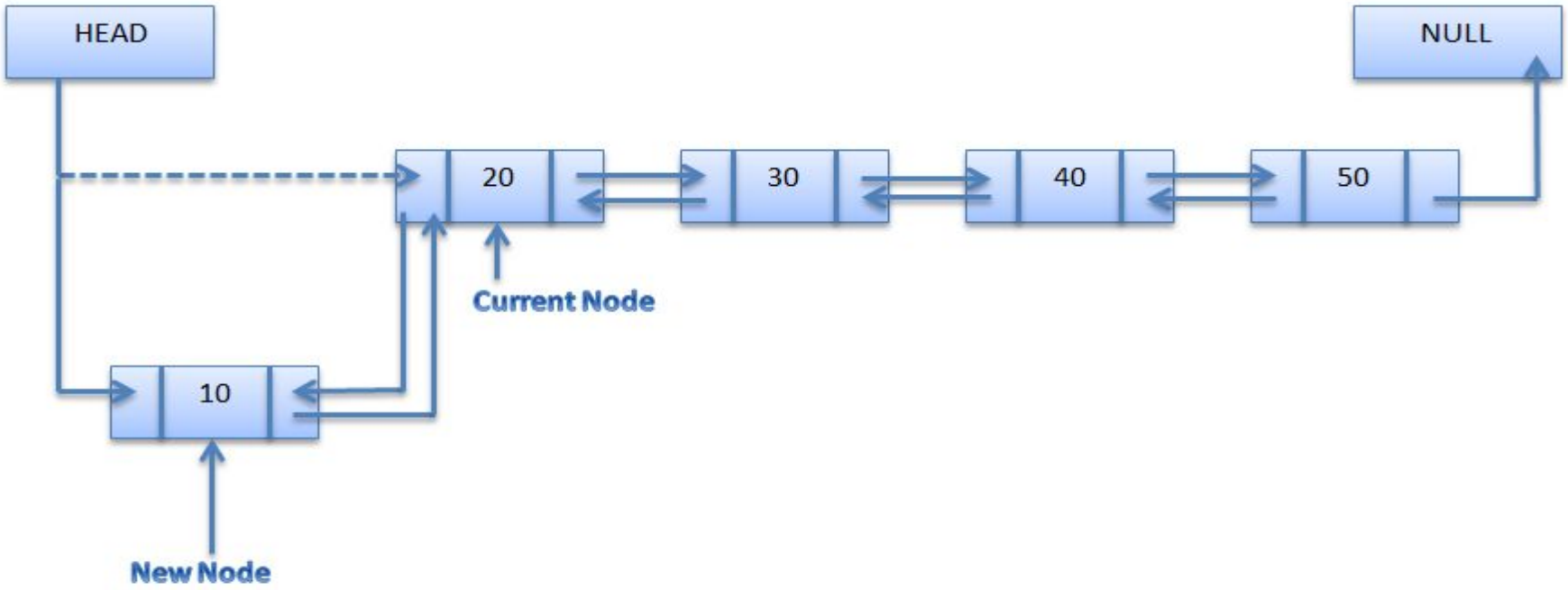
Creation and Traversal of doubly linked list (contd..)

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

```
# Singly Linked List with creation and
      print methods
LL = DoublyLinkedList()
LL.create(3)
LL.create(4)
LL.create(5)
LL.create(6)
LL.printLL()
```

Insertion at the Beginning Linked List

- Insertion at the Beginning of Doubly Linked List



Insertion at Beginning in Doubly linked list

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

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

```
# Insertion method for the doubly
  linked list at  beginning
    def insert_beg(self, data):
        newNode = Node(data)
        if(self.head==None):
            self.head = newNode

        else:
            newNode.next=self.head
            self.head.prev=newNode
            self.head=newNode
```

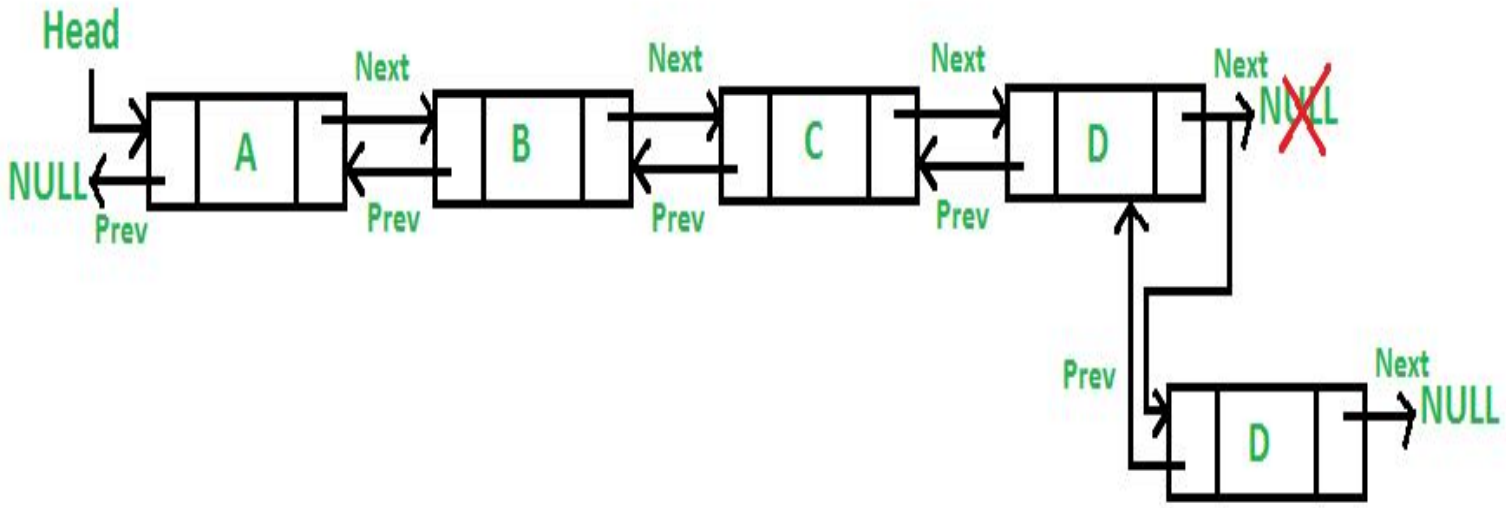
Insertion at Beginning in Doubly linked list (contd..)

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

```
# Singly Linked List with creation and
      print methods
LL = DoublyLinkedList()
LL.insert_beg(6)
LL.insert_beg(5)
LL.insert_beg(4)
LL.insert_beg(3)
LL.printLL()
```

Insertion at the end of Doubly Linked List

- Insertion at the end of Doubly Linked List



Insertion at end in Doubly linked list

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

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

```
# Insertion method for the doubly linked list at end
    def insert_end(self, data):
        newNode = Node(data)
        if(self.head==None):
            self.head = newNode

        else:
            temp=self.head
            while(temp.next!=None):
                temp=temp.next

            temp.next=newNode
            newNode.prev=temp
```

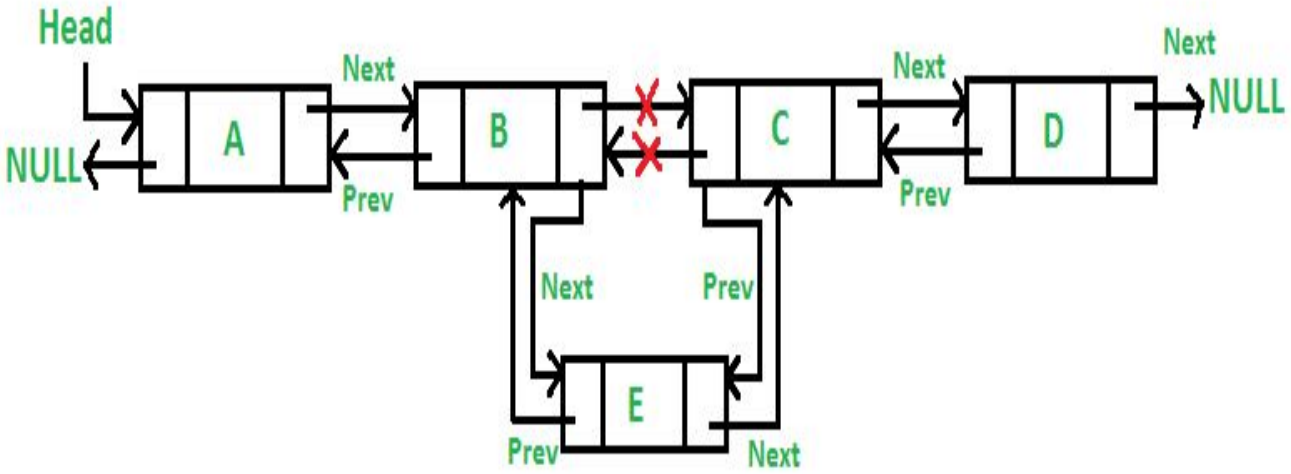
Insertion at end in Doubly linked list (contd..)

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

```
# Singly Linked List with creation and
      print methods
LL = DoublyLinkedList()
LL.insert_end(3)
LL.insert_end(4)
LL.insert_end(5)
LL.insert_end(6)
LL.printLL()
```

Insertion in Doubly Linked List (at position)

- Insertion at given position in Doubly Linked List



Insertion in Doubly Linked List (at position)

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


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

```
# creation method for the doubly
    linked list
    def create(self, data):
        newNode = Node(data)
        if(self.head==None):
            self.head = newNode

        else:
            temp=self.head
            while(temp.next!=None):
                temp=temp.next

            temp.next=newNode
            newNode.prev=temp
```

Insertion in Doubly Linked List (at position) (contd..)

```
# insertion method for the doubly
   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.prev=newNode
          current.next=newNode
          newNode.prev=current

       else:
          print("\nThe previous node is null.")
```

Insertion in Doubly Linked List (at position) (contd..)

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

```
# Singly Linked List with creation and
      print methods
LL = DoublyLinkedList()
LL.create(3)
LL.create(4)
LL.create(5)
LL.create(6)
LL.create(7)
LL.printLL()
LL.insert_position(4, 9)
LL.printLL()
```

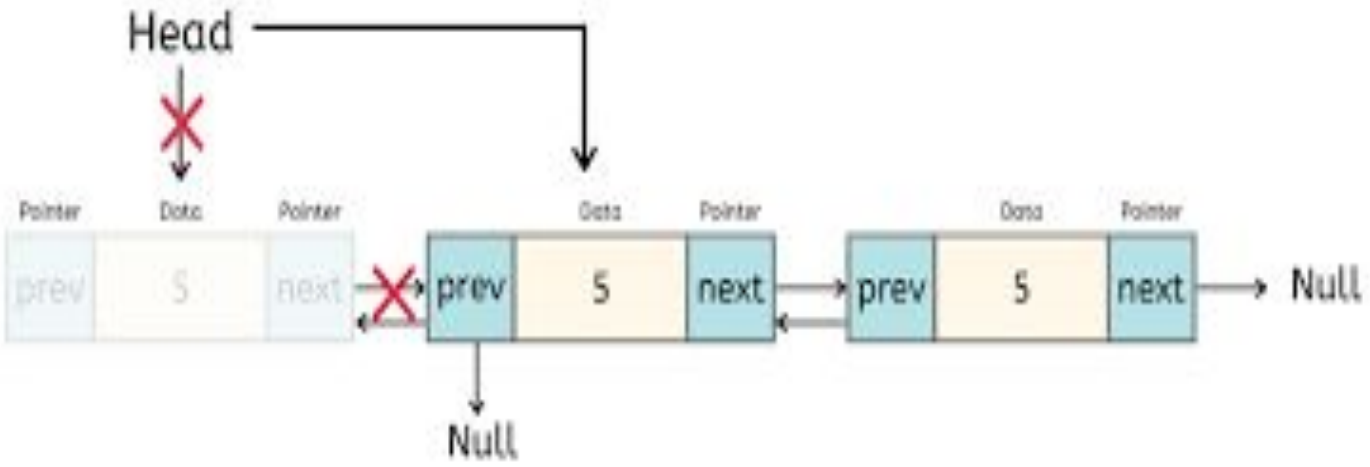
Deletion in a Doubly Linked List

- Similar to single linked list there are three possible positions where we can enter a new node in a doubly linked list –
  - **Deletion at beginning**
  - **Deletion at end**
  - **Deletion from given position**

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

Deletion in Doubly Linked List (from beginning)

- **Deletion from beginning**



Deletion in Doubly Linked List (from beginning)

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

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

```
# creation method for the doubly linked list
    def create(self, data):
        newNode = Node(data)
        if(self.head==None):
            self.head = newNode

        else:
            temp=self.head
            while(temp.next!=None):
                temp=temp.next

            temp.next=newNode
            newNode.prev=temp
```

Deletion in Doubly Linked List (from beginning) (contd..)

```
#Delete first node of the list
  def del_beg(self):
    if(self.head == None):
      print("Underflow-Link List is empty")
    else:
      temp = self.head
      self.head = self.head.next
      self.head.prev=None
      print("the deleted element is", temp.data)
      temp = None
```

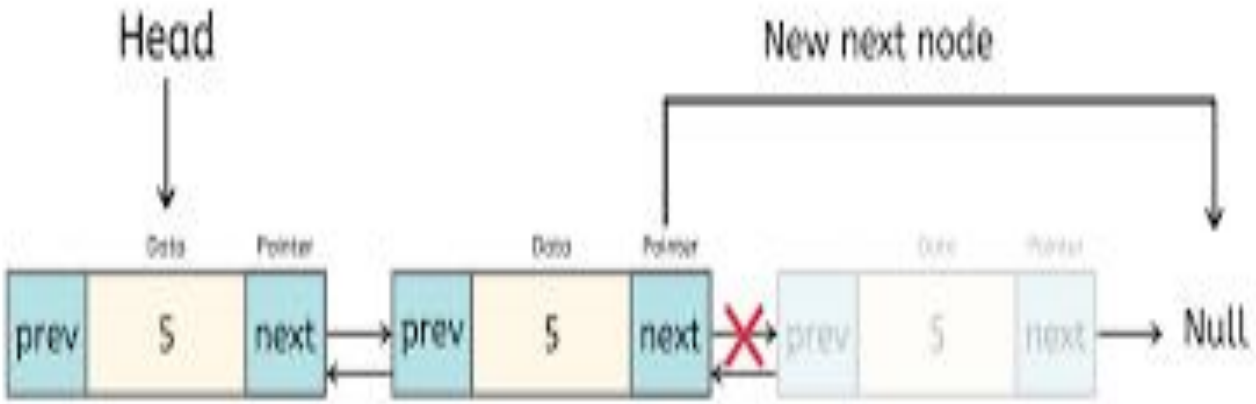
```
# print method for the linked list
  def printLL(self):
    current = self.head
    if(current!=None):
      print("The List
Contains:",end="\n")
      while(current!=None):
        print(current.data)
        current = current.next
    else:
      print("List is Empty.")
```

Deletion in Doubly Linked List (from beginning) (contd..)

```
# Doubly Linked List with creation, deletion and print methods
LL = DoublyLinkedList()
LL.create(3)
LL.create(4)
LL.create(5)
LL.create(6)
LL.printLL()
LL.del_beg()
LL.printLL()
```

Deletion in Doubly Linked List (from end)

- **Deletion from end**



Deletion in Doubly Linked List (from end)

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

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

```
# creation method for the doubly linked list
    def create(self, data):
        newNode = Node(data)
        if(self.head==None):
            self.head = newNode

        else:
            temp=self.head
            while(temp.next!=None):
                temp=temp.next

            temp.next=newNode
            newNode.prev=temp
```

Deletion in Doubly Linked List (from end) (contd..)

```
#Delete last node of the list
  def del_end(self):
    if(self.head == None):
      print("Underflow-Link List is empty")

    else:
     temp = self.head
     while(temp.next!=None):
        prev=temp
        temp=temp.next

     prev.next=None
     print("The deleted element is", temp.data)
     temp = None
```

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

Deletion in Doubly Linked List (from end) (contd..)

```
# Doubly Linked List with creation, deletion and print methods
LL = DoublyLinkedList()
LL.create(3)
LL.create(4)
LL.create(5)
LL.create(6)
LL.printLL()
LL.del_end()
LL.printLL()
```

Deletion in Doubly Linked List (from position)

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

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

```
# creation method for the doubly linked list
    def create(self, data):
        newNode = Node(data)
        if(self.head==None):
            self.head = newNode

        else:
            temp=self.head
            while(temp.next!=None):
                temp=temp.next

            temp.next=newNode
            newNode.prev=temp
```

Deletion in Doubly Linked List (from position) (contd..)

```
# Deletion method from the linked list at
  given position
  def del_position(self, pos):
     if(pos<1):
        print("\nPosition should be >=1.")


     else:
        temp=self.head
        for i in range(1, pos):
           if(temp!=None):
              current=temp
              temp=temp.next
        if(temp!=None):
           current.next=temp.next
           temp.next.prev=current
           print("the deleted element
is", temp.data)
           temp=None


        else:
           print("\nThe position does
not exist in link list.")
```

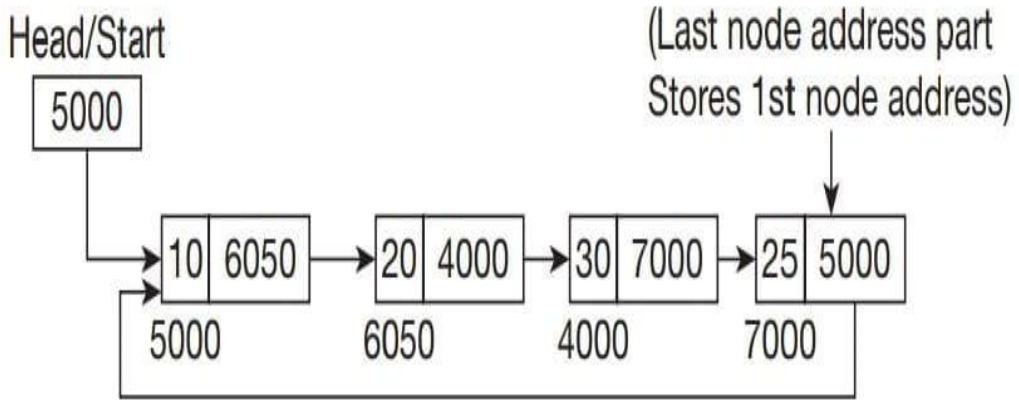
Deletion in Doubly Linked List (from position) (contd..)

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

```
# Doubly Linked List with creation,
      deletion and print methods
LL = DoublyLinkedList()
LL.create(3)
LL.create(4)
LL.create(5)
LL.create(6)
LL.create(7)
LL.create(8)
LL.printLL()
LL.del_position(4)
LL.printLL()
```

Circular Linked List:

In a singly linked list, each node points to its next node in the sequence, and the last node points to the null reference. In the circular linked list, every node points to its next node in the sequence, but the last node points to the first node in the list. Hence, the circular linked list is similar to the singly linked list except that the last node points to the first node in the list



Algorithm for inserting a node at the beginning of the circular linked list

Step 1: Create a Newnode with a specific value.
Step 2: Check whether list is empty (head = = NULL) or not.
Step 3: If it is empty, then set head = Newnode and
Newnode → next = head.
Step 4: If it is not empty, then define a node pointer 'temp' and initialize with 'head'.
Step 5: Keep moving the 'temp' to its next node until it reaches the last node (until 'temp → next == head').
Step 6: Set 'Newnode → next = head', 'head = Newnode' and 'temp → next = head'.

Algo for inserting a node at end of the circular linked list:

Step 1: Create a Newnode with a specific value.
Step 2: Check whether the list is empty (head == NULL) or not.
Step 3: If it is empty, then set head = Newnode and Newnode → next = head.
Step 4: If it is not empty, then define a node pointer temp and initialize with head.
Step 5: Keep moving the temp to its next node until it reaches the last node in the list (until temp → next == head).
Step 6: Set temp → next = Newnode and Newnode → next = head.

Polynomials

Polynomials are the algebraic expressions which consist of exponents and coefficients.

Example -
$10x^2 + 26x$, here 10 and 26 are coefficients and 2, 1 is its exponential value.

Polynomial can be represented in the various ways. These are:

- By the use of arrays
- By the use of Linked List

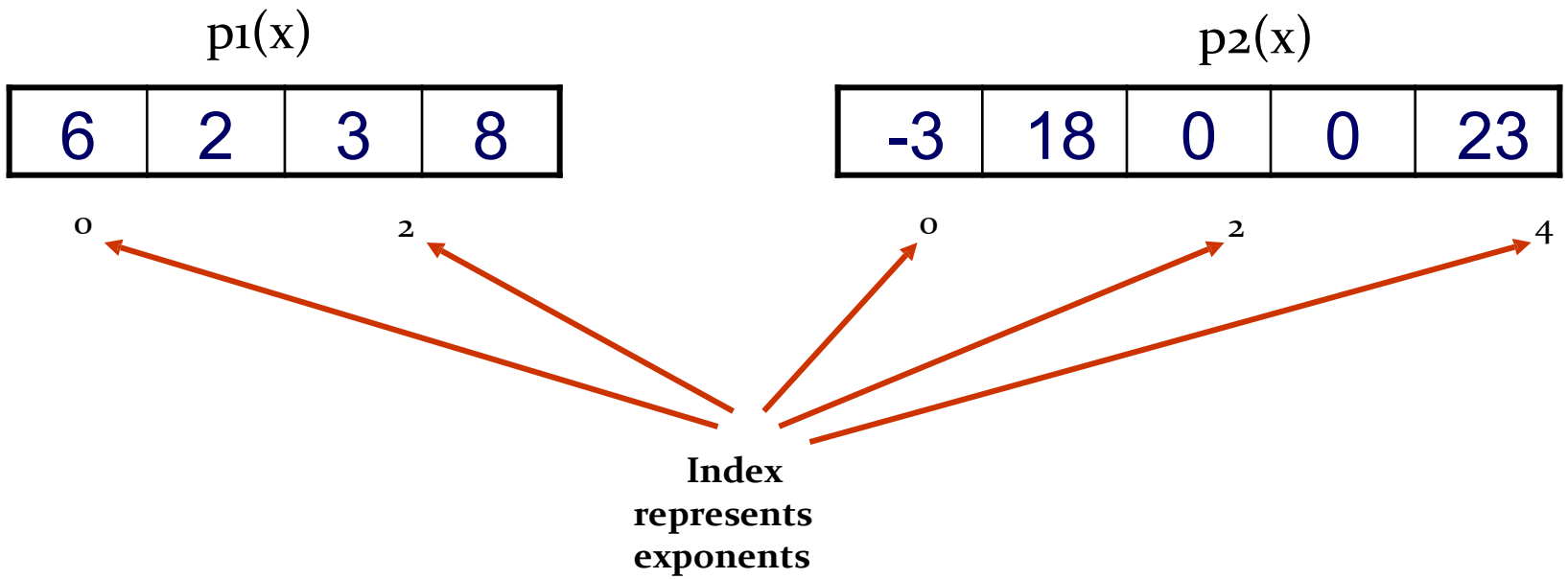
Polynomial Representation

Polynomial can be represented

- By the use of arrays
- By the use of Linked List

Polynomial (Array Representation)

- Array Representation:
  - $p1(x) = 8x^3 + 3x^2 + 2x + 6$
  - $p2(x) = 23x^4 + 18x - 3$



Polynomial (Array Representation)

- This is why arrays aren't good to represent polynomials:

- $p3(x) = 16x^{21} - 3x^5 + 2x + 6$

6	2	0	0	-3	0	.....	0	16

WASTE OF SPACE!

Add Two Polynomials Using Arrays

Input: A[] = {5, 0, 10, 6} B[] = {1, 2, 4}
Output: sum[] = {6, 2, 14, 6}

The first input array represents "5 + 0x^1 + 10x^2 + 6x^3“

The second array represents "1 + 2x^1 + 4x^2"

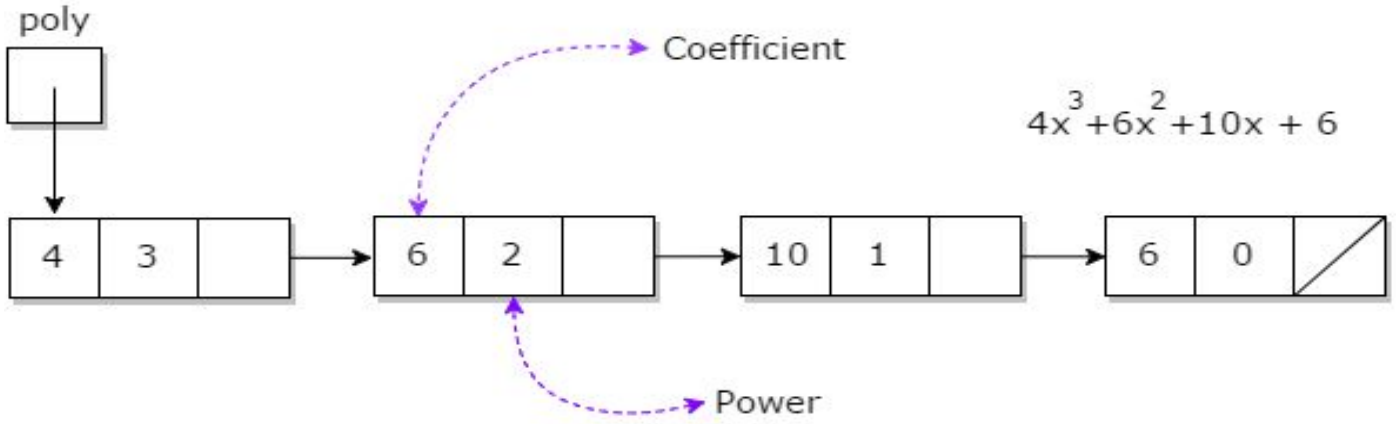
And Output is "6 + 2x^1 + 14x^2 + 6x^3“

Polynomial Representation (using Linked List)

- A polynomial p(x) is the expression in variable x which is in the form ($ax^n + bx^{n-1} + \ldots. + jx + k$), where a, b, c …., k fall in the category of real numbers and 'n' is non negative integer, which is called the degree of polynomial.
- An essential characteristic of the polynomial is that each term in the polynomial expression consists of two parts:
  - one is the coefficient
  - other is the exponent
- <u>Example:</u>
  - $10x^2 + 26x$,
    - here 10 and 26 are coefficients and 2, 1 is its exponential value.

Polynomial Representation (using Linked List)

- Points to keep in Mind while working with Polynomials:
- The sign of each coefficient and exponent is stored within the coefficient and the exponent itself
- Additional terms having equal exponent is possible one
- The storage allocation for each term in the polynomial must be done in ascending and descending order of their exponent



Polynomial Representation (using Linked List)

- Addition of polynomial

  1) Input: 1st number = $5x^2 + 4x^1 + 2x^0$

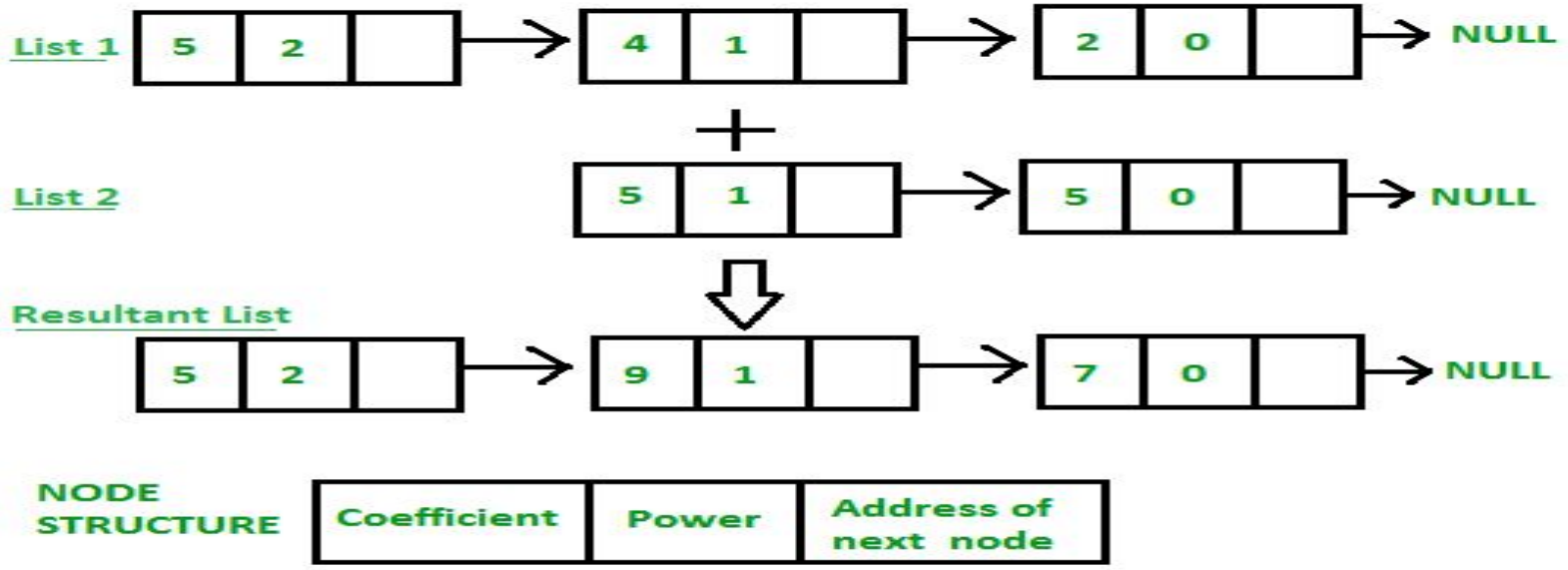
     2nd number = $-5x^1 - 5x^0$

     Output: $5x^2-1x^1-3x^0$

  2) Input: 1st number = $5x^3 + 4x^2 + 2x^0$

     2nd number = 5x^1 - 5x^0

     Output: $5x^3 + 4x^2 + 5x^1 - 3x^0$



Polynomial Representation (using Linked List)

Subtraction of polynomial

Input: 1st number = $5x^2 + 4x^1 + 2x^0$

2nd number = $-5x^1 - 5x^0$

Output: $5x^2 + 9x^1 + 7x^0$

Polynomial Representation (using Linked List)

Multiplication of polynomial

Input: Poly1: 3x^2 + 5x^1 + 6, Poly2: 6x^1 + 8

Output: 18x^3 + 54x^2 + 76x^1 + 48