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.

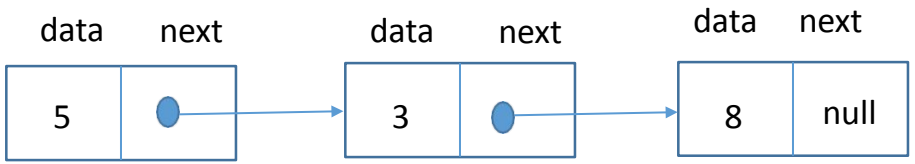
Arrays & Linked list

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]

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)

Node1

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()
```