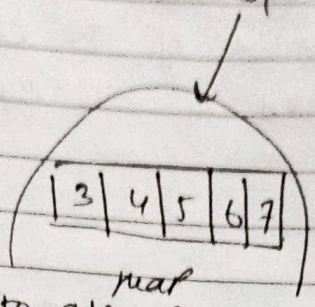
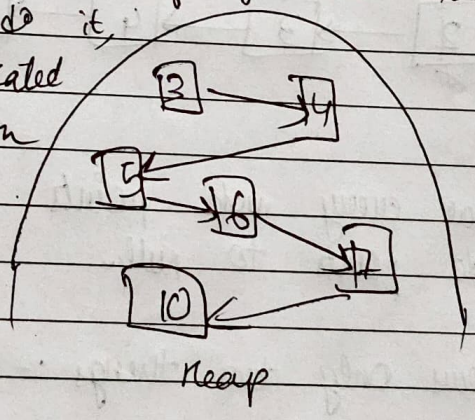
Linked Lists

While working with Arrays, we know that Arrays have contiguous memory allocation. That is it is stored something like this in heap. [3, 4, 5, 6, 7]



What if you want to add a new element to it. Of course, you can't add it in the array because its size is 5. Okay, one way is to arraylist. But, first you need to copy all the elements of the array to arraylist and then add further elements in it. This copying first and adding later is something not cool. That's when Linked List comes to play.

In Linked List, Random allocation in the memory for different components. So, in heap. If you want to add one more element, you can still do it, bc memory allocated is random. Say you want to add 10.



~~Singly Linked List~~

Components of Linked List

In a linked list, every element has two components -

① Data value.

② Pointer to the next node.

Both the components when put together, c/a Node.

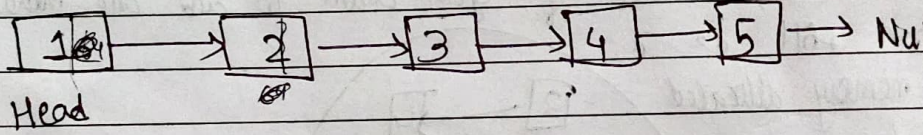
→ Data Value
It indicates what type of data will be stored in each element.

→ Pointer to the next Node –
The next component of the element will have a pointer to the address of the next element in the linked list.
It indicates

Note 1. There is a pointer which points to the starting node of the linked list. This pointer is known as HEAD.
2. Sometimes, there is also a pointer which points towards the last node of the linked list, c/a TAIL
3. The next pointer of the last node points towards NULL
4. NULL means that the pointer is not pointing to anything

3 types

① SINGLY LINKED LIST



Ye kuch kai, that every node points to the next element and last node points to null.

Every node knows only two things :- it's value, and the next element it is pointing to.

Think of it as, "1 loves 2, 2 loves 3, 3 loves 4, 4 loves 5. but 2 has no idea that 1 loves it, 3 has not idea that 2 loves it, 4 has no idea that 3 loves it and 5 has no idea that 4 loves it". It only knows itself and the number it loves.

Structure of the node.

```
class Node {
    int val;
    Node next;
}
```

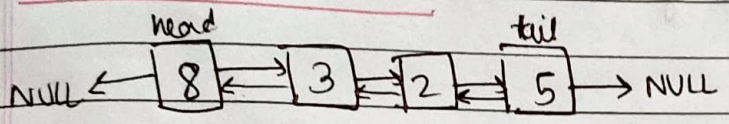
One Node has two things, one type of itself i.e int in this type and value of next element which is also of type Node w/o same properties.

→ LinkedList<Integer> list = new LinkedList<>();
list.add(3);

You can use these.

→ But, we can also make our own Linked List.
{Check out 'codes' section for various operations on Linked List}

② # DOUBLY LINKED LIST



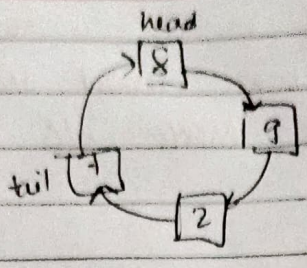
1. In a doubly linked list, there are two pointers.
2. The first pointer is the next pointer which points to the next node in the list.
3. The second pointer is the previous pointer which points to the previous node in the list.
4. As usual, there will only be one data value in the node.
5. The next pointer of the last node will be pointed towards NULL.
6. The previous pointer of the first node will be pointed towards NULL.

Structure

```
class Node {
    int val;
    Node next;
    Node prev;
}
```

{Check out codes for various operations on Doubly Linked List}

③ # Circular Linked List




```
class Node{
int val;
Node next;
};
```

1. Circular linked list is same as singly linked list with just one difference. It is that the last node of the linked list (tail) points towards the first (head) forming a circular cycle.
2. No node in the linked list points towards NULL.
3. If we start from any node, then after traversal we will again come back to the same node.
4. Both, the last node of the linked list and head pointer points towards the start node of the linked list.