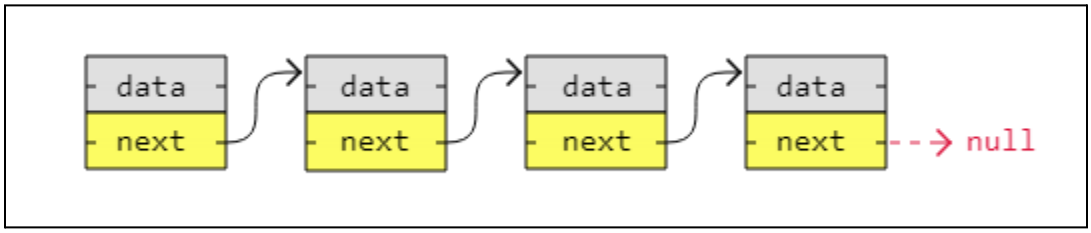
LikedList:-

We already know that the data in **Array** is stored in contiguous address spots inside the memory. But what if we don't have enough contiguous space to store many elements together? To overcome this problem, we define another data structure called **Linked List**.

A linked list consists of nodes with some sort of data, and a pointer, or link, to the next node.



A big benefit of using linked lists is that nodes are stored wherever there is free space in memory, the nodes do not have to be stored contiguously right after each other like elements are stored in arrays. Another nice thing with linked lists is that the rest of the nodes in the list do not have to be shifted when adding or removing nodes.

Difference between Array and Linked List

Factor	Array	LinkedList
Memory Allocation	Contiguous allocation in the memory for different components	Random allocation in the memory for different components.
Memory Type	It is static. Uses static memory. Eg, Stack memory	It is dynamic. Uses dynamic memory. Eg, Heap memory.
Accessing	Direct access is there. If we want to find the value of the 5th element, We can just find arr[4]. Time Complexity - O(1) constant	Indirect access is there. If we want to find the value of the 5th element, we need to traverse the linked list up to the 5th element. Time Complexity - O(N) linear
Dependency	The elements of the array are independent of each other.	The current element of the linked list is dependent on the previous element of the linked list.

Insertion	It is difficult to add any element at a particular position of the array. We may not find an external address just right to the current array. **Time complexity - O(N)**	It is easy to add any element in the linked list at any position. We don't need to shift or modify the linked list. **Time complexity -** **Insertion at start - O(1)** **Insertion not at start - O(N)**
Deletion	It is difficult to delete any element from the array. First, we need to shift all the right elements by one unit to the left. Also, the last position will remain empty and space will be wasted. **Time Complexity - O(N)**	It is easy to delete an element from the linked list. No space is wasted. No need to change or modify the linked list. **Time Complexity -** **Deletion at start - O(1)** **Deletion not at start - O(N)**

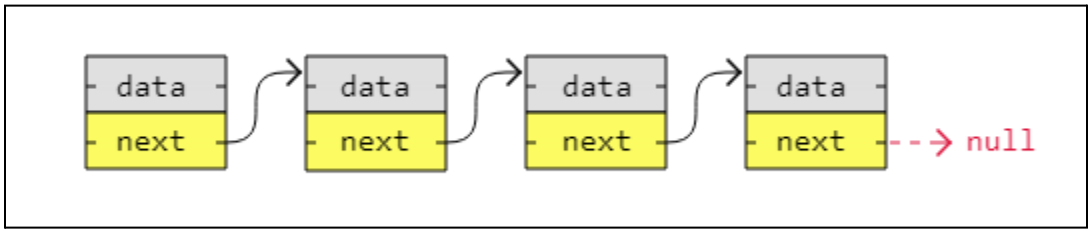
Types of Linked List

There are 3 types of linked list -

1. **Singly Linked List**
2. **Circular Linked List**
3. **Doubly Linked List**

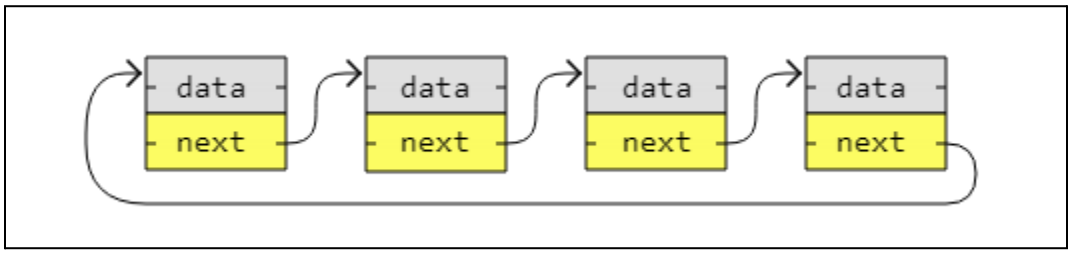
Singly Linked List

1. In a singly linked list, there is only **one pointer** which points to the **next node** of the list.
2. As usual, there will only be **one data** value in the node.
3. The last node points towards **NULL**.
4. The **HEAD** points towards the first node.



Circular Linked List

1. Circular Linked list is the same as the singly linked list with just one difference. It is that the last node of the linked list **(tail)** points towards the first node **(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.



Doubly Linked List

1. In a doubly linked list, there are **two** pointers.
2. The first pointer is called the **next pointer** which points to the next node in the list.
3. The second pointer is called 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.