# Linked Lists

## Introduction

- A linked list is a **linear data structure** where each element is a separate object.
- Each element of a linked list is called a **node**.
- Each node of a list is made up of two items - the **data** and reference (also known as **pointer**) to the next node.
- Each linked list needs a pointer to the first node of the list. This pointer is known as **head**.

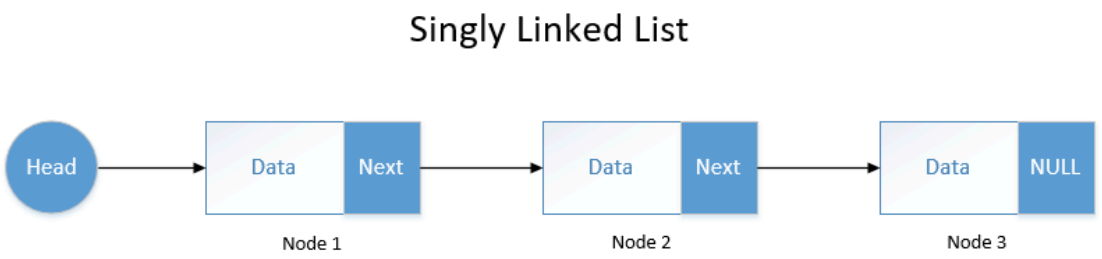
## Types of Linked Lists

There are three kinds of linked lists depending on how the nodes are linked to each other:

- **Singly Linked List**
- **Doubly Linked List**
- **Circular Linked List**

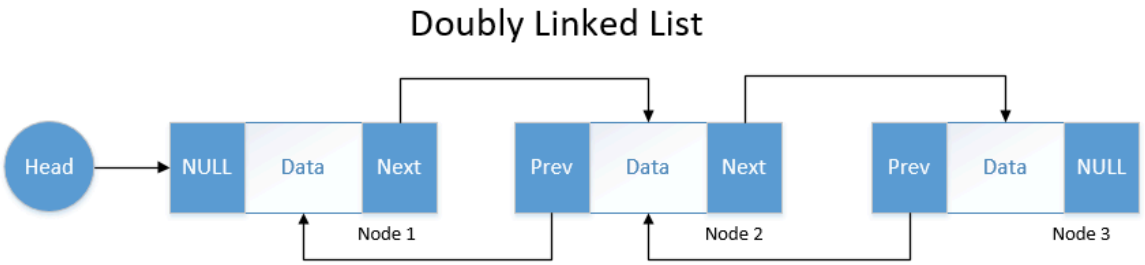
Open each section below to know more.

### 1 Singly Linked List




- Each node has one pointer called **Next** which stores address of the next node.
- A node which doesn't point to any other node is known as **Tail**.
- Traversal can happen only in **forward** direction since each node has only information of the node after it.

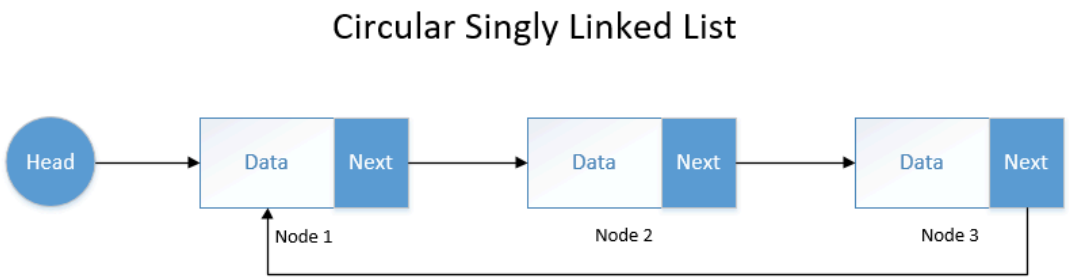
## 2 Doubly Linked List




- Each node has two pointers called **Prev** and **Next** which stores address of the previous and next node respectively.
- The first node has no node before it, so Prev is NULL here.
- The last node has no node after it, so Next is NULL here.
- Traversal can happen both in **forward** and **backward** directions since each node has information of the node before and after it.

## 3 Circular Linked List

Both Singly and Doubly Linked Lists can be made circular by just connecting the first and last node.




To convert a singly linked list into circular:

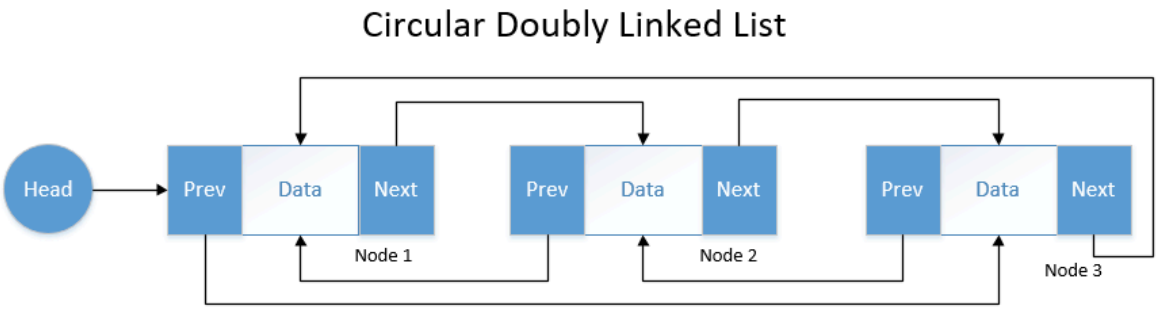
The last node has the first node as the node just after it, so Next should point to first node.

**Note:**

Observe the above version of Singly Linked List. Instead of having a tail node, Node 3 has a pointer to Node 1.

Looking at the direction of arrows you will realize that there's a cycle:

Node 1 => Node 2 => Node 3 => Node 1 => Node 2 => Node 3 => Node 1 => and so on... (Hence Circular!)




To convert a doubly linked list into circular:

- The first node has the last node as the node just before it, hence Prev should point to last node.
- The last node has the first node as the node just before it, hence Next should point to first node.

Pro Tip: **Linked Lists v/s Arrays**

## How to code linked lists?

You might have found your inclination to one of the programming languages by now. It can be C/C++/Java/Python etc.

Depending on your choice you can go through the below articles to find examples for the same. Remember it's about finding that right pair of shoes to get going with coding. Don't restrict yourself.

1. Basic Implementation: https://www.geeksforgeeks.org/linked-list-set-1-introduction/
2. Insertion of a node: https://www.geeksforgeeks.org/linked-list-set-2-inserting-a-node/
3. Deletion of a node: https://www.geeksforgeeks.org/delete-a-linked-list-node-at-a-given-position/
4. Searching: https://www.geeksforgeeks.org/search-an-element-in-a-linked-list-iterative-and-recursive/
5. Swapping nodes: https://www.geeksforgeeks.org/pairwise-swap-elements-of-a-given-linked-list/

# Additional Reads (Optional)

If you have time left go through the following links for more interesting applications and concepts:

- https://www.geeksforgeeks.org/how-does-floyds-slow-and-fast-pointers-approach-work/
- https://www.geeksforgeeks.org/function-to-check-if-a-singly-linked-list-is-palindrome/
- https://www.geeksforgeeks.org/pairwise-swap-elements-of-a-given-linked-list/
- https://www.geeksforgeeks.org/josephus-circle-using-circular-linked-list/
- https://www.geeksforgeeks.org/swap-kth-node-from-beginning-with-kth-node-from-end-in-a-linked-list/
- https://www.geeksforgeeks.org/remove-duplicates-sorted-doubly-linked-list/

Exhaustive list : https://www.geeksforgeeks.org/data-structures/linked-list/