Linked List:

A Linked List is a linear data structure where each element, called a node, points to the next node in the sequence. Unlike arrays, linked lists do not store elements in contiguous memory locations, making it easier to insert or delete elements without reallocation or reorganization of the entire structure.

Basic Concepts:

- **Node:** A basic unit of a linked list containing two parts:
  - Data: The value stored in the node.
  - Next: A reference (or pointer) to the next node in the list.
- **Head:** The first node in the linked list.
- **Tail:** The last node in the linked list, which usually points to `null`.

Types of Linked Lists:

- **Singly Linked List:** Each node points to the next node, and the last node points to `null`.
- **Doubly Linked List:** Each node has two pointers, one to the next node and one to the previous node.
- **Circular Linked List:** The last node points back to the first node, forming a circle.

Implementation of Singly Linked List:

```
class Node{
   constructor(data){
      this.data=data;
      this.next=null;
   }
}

var head=new Node(1);
var second=new Node(20);
var third=new Node(30);

head.next=second;
second.next=third;
```

Operations Performed:

1.Printing a Singly Linked List:

```
var ptr=head;
while(ptr!=null){
    console.log(ptr.data);
    ptr=ptr.next;
}
```

2.Count the number of nodes in a Linked List:

```
var count=0;
var ptr=head;
while(ptr!=null){
      count++;
      ptr=ptr.next;
}
```

3.Addition of a node:

I. Adding in the beginning of Linked List:

```
var newNode=new Node(100);
newNode.next=head;
head=newNode;
```

II. **Adding in the end of Linked List:**

```
var p=head;
while(p.next!=null){
    p=p.next;
}

var newNode=new Node(100);
p.next=newNode;
```

III.Adding at a specific position:

```
var p=head;
var pos=2;
var i=1;
while(i<=pos-1){
    p=p.next;
    i++;
}
```

```
var newNode=new Node(100);
newNode.next=p.next;
p.next=newNode;
```

4.Deletion of a node:

I.Deletion in the beginning:

```
var newHead=head.next;
head.next=null;
head=newHead;
```

II.Deletion at the end:

```
var p=head;
while(p.next.next!=null){
    p=p.next;
}

p.next=null;
```

III.Delete at a specific position:

```
var p=head;
var pos=1;
var i=0;
while(i<=pos-1){
    p=p.next;
    i++;
}
var nextnode=p.next;
p.next=nextnode.next;
nextnode.next=null;
```

Practice Problems:

1. Reverse a Linked List:

```
var reverseList = function(head) {
    var next=null;
    var prev=null;
    var curr=head;
    while(curr!=null){
        next=curr.next;
        curr.next=prev;
        prev=curr;
        curr=next;
    }
    return prev;
};
```

2.Middle Of the Linked List:

```
var middleNode = function(head) {
    var slow=head;
    var fast=head;
    while(fast!=null&&fast.next!=null){
        slow=slow.next;
        fast=fast.next.next;
    }
    return slow;
};
```

3.Check if linked list is Palindrome or not.
Problem
Link:https://leetcode.com/problems/palindrome-linked-list/description/

Solution:

```
function middleOfLinkedList(head){
    var slow=head;
    var fast=head;
    while(fast.next!=null&&fast.next.next!=null){
        slow=slow.next;
        fast=fast.next.next;
    }
    return slow;
}
function printList(mid){
    var ptr=mid;
while(ptr!=null){
    console.log(ptr.val);
    ptr=ptr.next;
}
}
function reverseALinkedList(newHead){
    var curr=newHead;
    var prev=null;
    var next=null;
    while(curr!=null){
        next=curr.next;
        curr.next=prev;
        prev=curr;
        curr=next;
    }
    return prev;
}
function isPalindromic(head,head2){
    while(head!=null&&head2!=null){
        if(head.val!=head2.val){
            return false;
        }

        head=head.next;
        head2=head2.next;
    }
    return true;
}
var isPalindrome = function(head) {
    if(head==null||head.next==null){
        return true;
    }
    var mid=middleOfLinkedList(head);
    var newHead=mid.next;
    mid.next=null;
    var head2=reverseALinkedList(newHead);
    var ans=isPalindromic(head,head2);
    return ans;
};
```