

ChatGPT said:







What is a Linked List in JavaScript?

A **Linked List** is a **linear data structure** where elements (called **nodes**) are stored in **separate objects**, and **each node points to the next node** in the sequence.

Unlike arrays:

- Linked lists **do not use indexes**.
- Data is stored **non-contiguously** in memory.
- You can **insert and delete efficiently** (especially at the beginning or middle).

Why Use Linked List Instead of Array?

Feature	Array	Linked List
Memory	Contiguous	Non-contiguous
Insert at beginning	 Costly ($O(n)$)	 Fast ($O(1)$)
Delete from middle	 Costly ($O(n)$)	 Efficient ($O(1)$ if pointer)
Access by index	 $O(1)$	 $O(n)$

Structure of a Linked List Node

Each node has:

1. **data**: value stored.
2. **next**: pointer to the next node (or **null** if it's the last node).

js

CopyEdit

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

```
}  
}
```

Full Example: Singly Linked List Implementation

js

CopyEdit

```
class Node {  
  constructor(data) {  
    this.data = data;  
    this.next = null;  
  }  
}  
  
class LinkedList {  
  constructor() {  
    this.head = null;  
  }  
  
  // Add at end  
  append(data) {  
    const newNode = new Node(data);  
    if (!this.head) {  
      this.head = newNode;  
      return;  
    }  
  
    let current = this.head;  
    while (current.next !== null) {  
      current = current.next;  
    }  
  
    current.next = newNode;  
  }  
  
  // Add at start  
  prepend(data) {  
    const newNode = new Node(data);  
    newNode.next = this.head;  
    this.head = newNode;  
  }  
}
```

```

    }

    // Print all nodes
    printList() {
        let current = this.head;
        let output = "";
        while (current !== null) {
            output += current.data + " -> ";
            current = current.next;
        }
        console.log(output + "null");
    }

    // Delete a node by value
    delete(data) {
        if (!this.head) return;

        if (this.head.data === data) {
            this.head = this.head.next;
            return;
        }

        let current = this.head;
        while (current.next && current.next.data !== data) {
            current = current.next;
        }

        if (current.next) {
            current.next = current.next.next;
        }
    }
}

```

Example Usage

```

js
CopyEdit
const list = new LinkedList();

list.append(10);

```

```
list.append(20);
list.append(30);
list.printList(); // 10 -> 20 -> 30 -> null

list.prepend(5);
list.printList(); // 5 -> 10 -> 20 -> 30 -> null

list.delete(20);
list.printList(); // 5 -> 10 -> 30 -> null
```

Time & Space Complexity

Operation	Time Complexity	Space Complexity
Insert (start)	O(1)	O(1)
Insert (end)	O(n)	O(1)
Delete	O(n)	O(1)
Search	O(n)	O(1)

Types of Linked Lists

1. **Singly Linked List** → Each node points to the next node only.
 2. **Doubly Linked List** → Each node points to both **next** and **prev**.
 3. **Circular Linked List** → The last node points back to the first node.
-

Doubly Linked List Mini Example

```
js
CopyEdit
class DoublyNode {
  constructor(data) {
    this.data = data;
    this.prev = null;
```

```

    this.next = null;
  }
}

```

Let me know if you want full code for doubly or circular linked lists too.

Summary

- Linked lists are made of nodes connected using `next`.
- They're useful for frequent insert/delete operations.
- Accessing a specific item is slower ($O(n)$) compared to arrays ($O(1)$).
- JavaScript doesn't have a built-in linked list, but we can build it using classes.

Would you like practice questions or visual diagrams next?

You said:

yes visual diagram

ChatGPT said:

Great! Here's a **visual diagram** of a **Singly Linked List** in JavaScript, followed by a simple step-by-step flow of how nodes are added or removed.

Singly Linked List Visual Diagram

Structure of Nodes

sql

CopyEdit

```

+-----+      +-----+      +-----+      +-----+
|  Data  | ---> |  Data  | ---> |  Data  | ---> | null |
|  (10)  |      |  (20)  |      |  (30)  |      |      |
+-----+      +-----+      +-----+      +-----+
      head                                     tail (null)

```

Each node has:

- **data**: The value stored (e.g., 10, 20, 30)
 - **next**: A pointer to the next node (or **null** if it's the last)
-

append(10) → append(20) → append(30) Flow

Step 1: Append 10

js

CopyEdit

```
list.append(10);
```

csharp

CopyEdit

```
Head → [10 | null]
```

Step 2: Append 20

js

CopyEdit

```
list.append(20);
```

css

CopyEdit

```
Head → [10 | o-] → [20 | null]
```

Step 3: Append 30

js

CopyEdit

```
list.append(30);
```

css

CopyEdit

```
Head → [10 | o-] → [20 | o-] → [30 | null]
```

Deleting a Node (e.g., delete(20))

Before:

css

CopyEdit

Head → [10 | o-] → [20 | o-] → [30 | null]

After deleting 20:

css

CopyEdit

Head → [10 | o-] → [30 | null]

The **next** pointer of node 10 now skips over 20 and directly points to 30.

Linked List vs Array Memory

Array (Contiguous Memory)

scss

CopyEdit

[10] [20] [30] → fixed in memory

↑ ↑ ↑
index 0 1 2

Linked List (Non-Contiguous Memory)

css

CopyEdit

[10 | o-] → [20 | o-] → [30 | null]
 @001 @129 @085

Each node is stored at **random memory addresses**, linked by **next** pointers.