# Linear Data Structures Using Node.js

On the fundamentals of Data Structures & Algorithms (DSAs) in JavaScript for those who don't know where to start.

## Things You'll Want:

1. Node.js
   a. You already have this installed if you use React.
2. Visual Studio Code Extension: Code Runner
   a. Install this so we can work within the confines of VS Code using Node.js.

## PERSIST, PERSIST, AND PERSIST

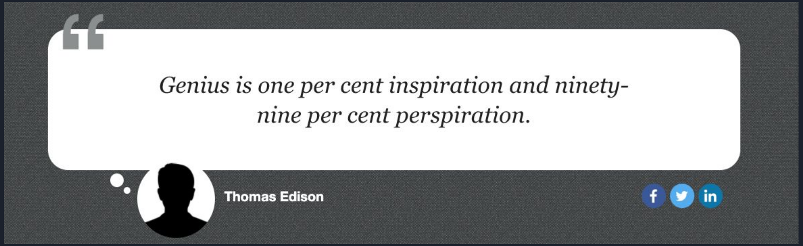
Begin at once and stay focused.

Track your learning.

Track your setbacks.

Track your challenges.

Track your wins.



*Genius is one per cent inspiration and ninety-nine per cent perspiration.*

Thomas Edison

# TRACK, TRACK, TRACK

Create a spreadsheet (e.g., Microsoft Excel, Google Sheets, etc.) for Data Structures with the following as Column Headers (or feel free to make up your own):

**Subject**
**Daily Data Structure Implemented?**
**Pomodoros**
**Source**
**Date**

## On Learning...

- Yes, you weren’t taught how to learn correctly
    - (especially if you’re American, like me 😆 )
- DSA/Coding/React is tough, so it pays massive dividends to learn *how to learn*.
- Is there a solution?

## A few solutions

- Learn how to *Actively Recall*.
- Learn about *Spaced Repetition*.
- Combining these two techniques will ensure you remember that which you need to make stick.

## Active Recall

Pause your studying.

Close your eyes (optional).

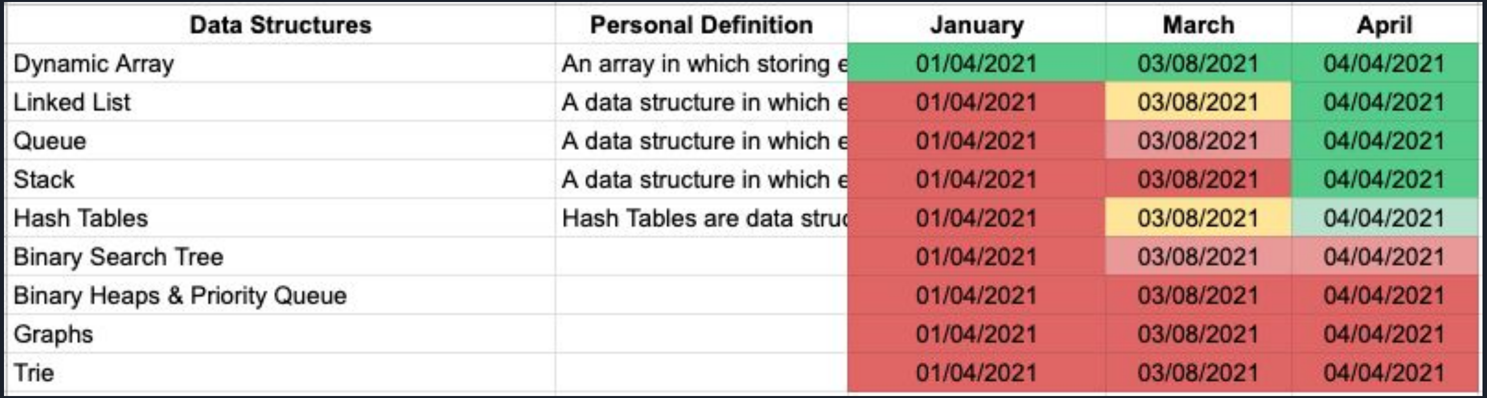
Talk out loud (optional) about that which you’ve just learned.

You are *actively recalling* the very thing you just learned about.

Continue periodically remembering that information so as to interrupt forgetting (this is known as Spaced Repetition).

# Spaced Repetition

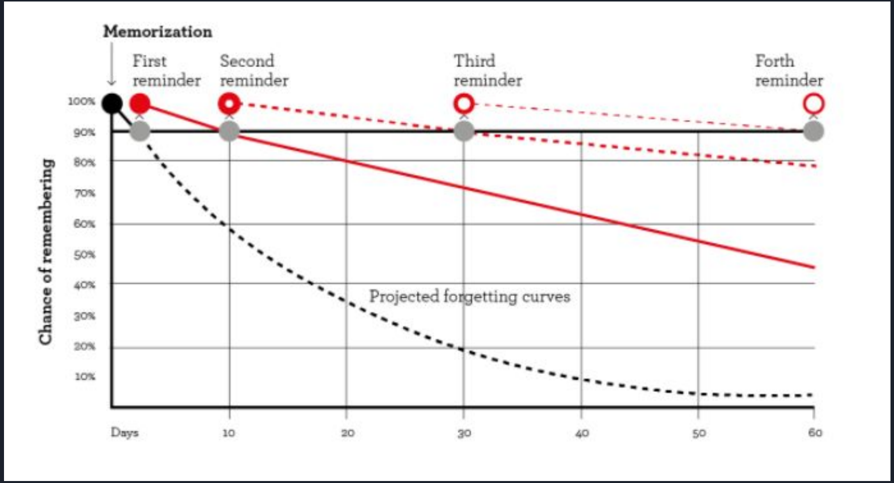
- Make your learning a function of time, not topics.
- The highlighted dates indicate understanding.



| Data Structures | Personal Definition | January | March | April |
|---|---|---|---|---|
| Dynamic Array | An array in which storing e | 01/04/2021 | 03/08/2021 | 04/04/2021 |
| Linked List | A data structure in which e | 01/04/2021 | 03/08/2021 | 04/04/2021 |
| Queue | A data structure in which e | 01/04/2021 | 03/08/2021 | 04/04/2021 |
| Stack | A data structure in which e | 01/04/2021 | 03/08/2021 | 04/04/2021 |
| Hash Tables | Hash Tables are data struc | 01/04/2021 | 03/08/2021 | 04/04/2021 |
| Binary Search Tree | | 01/04/2021 | 03/08/2021 | 04/04/2021 |
| Binary Heaps & Priority Queue | | 01/04/2021 | 03/08/2021 | 04/04/2021 |
| Graphs | | 01/04/2021 | 03/08/2021 | 04/04/2021 |
| Trie | | 01/04/2021 | 03/08/2021 | 04/04/2021 |

## Spaced Repetition

Through the application of *Spaced Repetition*, a learner may interrupt the forgetting curve:



## New Mindset

- Data Structures and Algorithms are initially very foreign to developers when they begin learning them, which is natural.
- However, that feeling will pass soon enough.
- Arrays and Objects are data structures, and you use them ALL the time.
- You give them no second thought because they are essential data structures we use on a daily basis.
- I want you to think of Data Structures as allies, not enemies.

## New Mindset Continued....

You will use:

- While Loops
- ES6 Classes
- Constructors & Methods
- The *this* keyword 🥲😁
- Setters and Getters

## Wait, all that sounds incredibly useless.

- And you’re not totally wrong in thinking that.
  - I mean, who uses a while loop in their daily coding?
- But those things WILL make you a better developer
  - Especially if you work in React.js

## Wait, it will make me a better React Developer? 👂

Yes.

Prior to the release of React Hooks in 2019, React used class-based components.

For companies who have written their entire codebase prior to 2019, guess what?

Exactly, they used class-based components, something you really ought to know if you want to become a React Developer.

How many companies have their source code written using React Hooks ONLY?

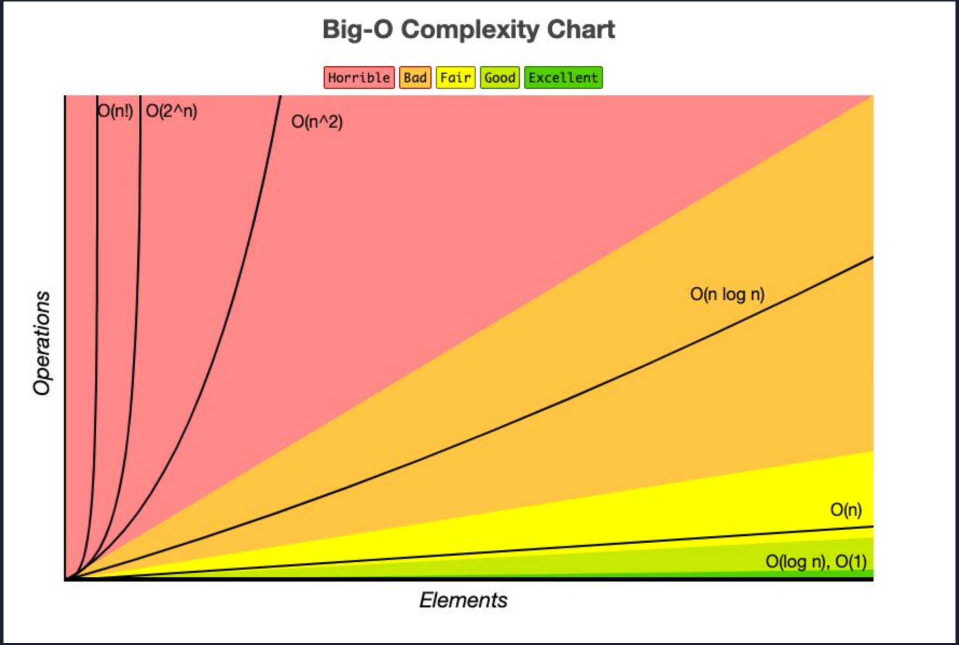
Probably a very small number of companies who

a) Are younger than the release of React Hooks in 2019, or
b) Have incredible resources to pay their React Developers to switch out already functioning class-based components to functional components.

So let's think about it. There's probably wayyyyyyy more class-based components in the world than functional hooks.

You should probably get a solid understanding of ES6 Classes, which will ultimately make you a better React Developer.

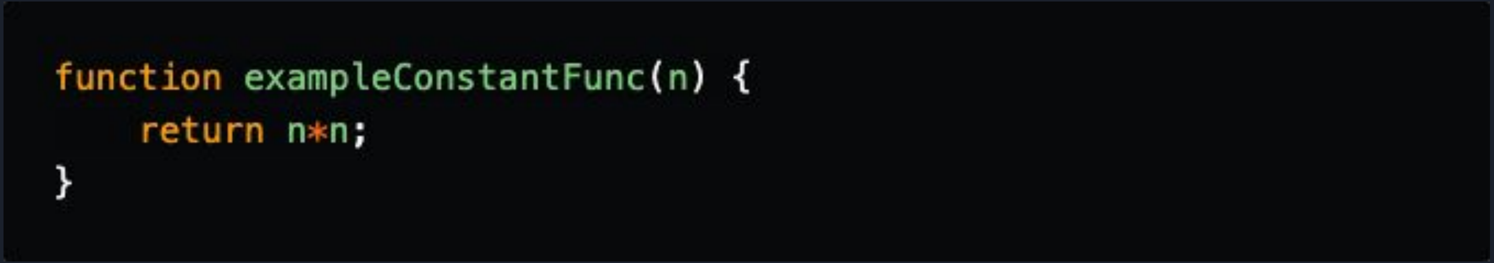
# Big O Notation



## Constant Runtime: Big O(1)

As input size *n* grows, a limited number of operations is used.

In this example, no matter how large our argument grows, only one operation is performed.

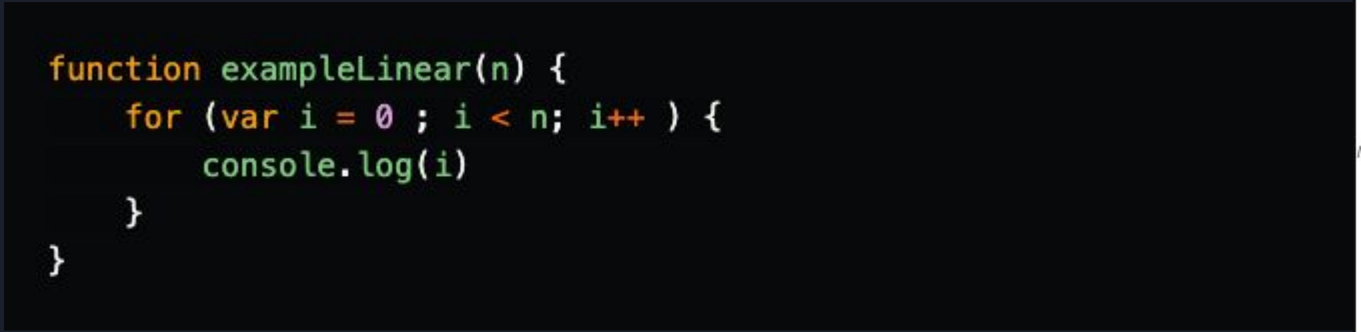


```
function exampleConstantFunc(n) {
    return n*n;
}
```

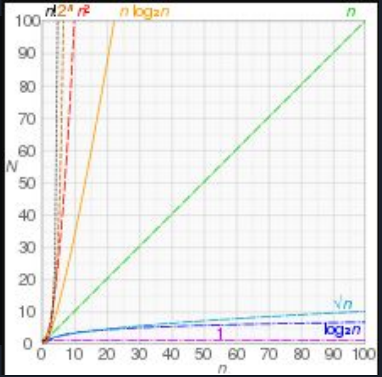
## Big O(n): Linear Runtime

As input size *n* grows, the number of operations grow in direct proportion relative to the size of *n*.

In this example, the number of operations grow exactly as our input size grows.



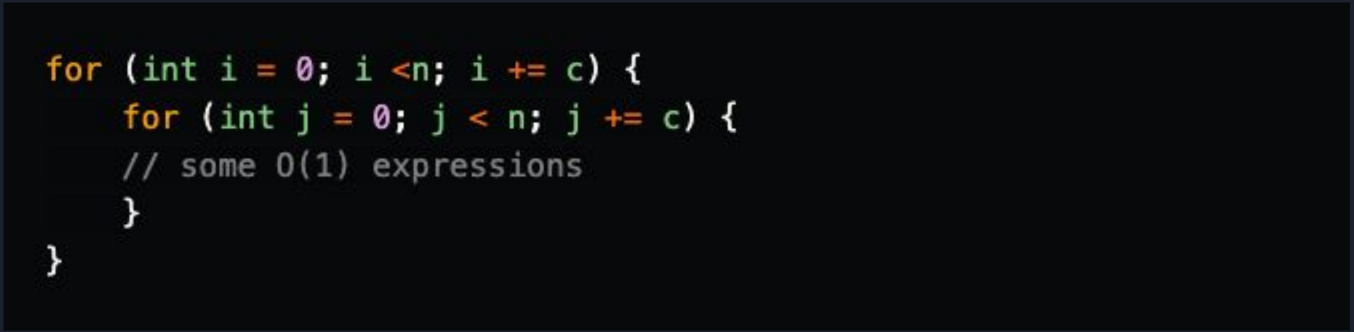
```
function exampleLinear(n) {
    for (var i = 0 ; i < n; i++ ) {
        console.log(i)
    }
}
```



# O($n^2$): Quadratic Runtime

As input size *n* grows, the number of operations grows quadratically (*n* raised to the power of 2).

In this example, the number of operations is $n^2$ because of the nested for loop.



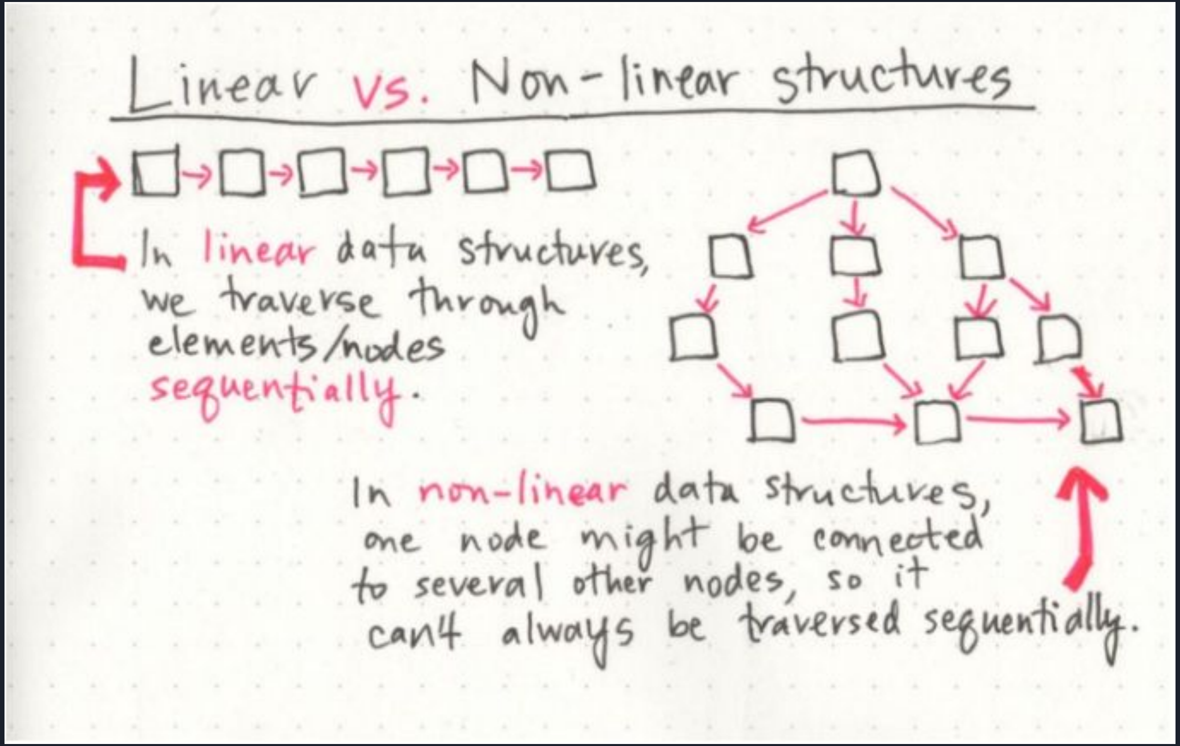
```
for (int i = 0; i <n; i += c) {
    for (int j = 0; j < n; j += c) {
    // some O(1) expressions
    }
}
```

## Input Size *n* vs Number of Operations

- As input size *n* grows,
    - Does the number of operations stay the same?
        - For example, an input *n* size of 100 has only one or two operations.
            - In an array of 100, accessing one element only.
    - Do the number of operations grow linearly relative to the size of the input *n*?
        - For example, an input *n* size of 100 means 100 operations.
            - Think of this in terms of a for-loop!
    - Do the number of operations grow logarithmically, quadratically, exponentially, factorially relative to the input size *n*?
        - What are the implications of a nested for loop? How about two nested for loops?

- Big O indicates the worst-case runtime of an algorithm
- Big O pertains to both *Time Complexity* and *Space Complexity*.
- The runtime of an algorithm is measured by its number of operations relative to the input size, *n*.

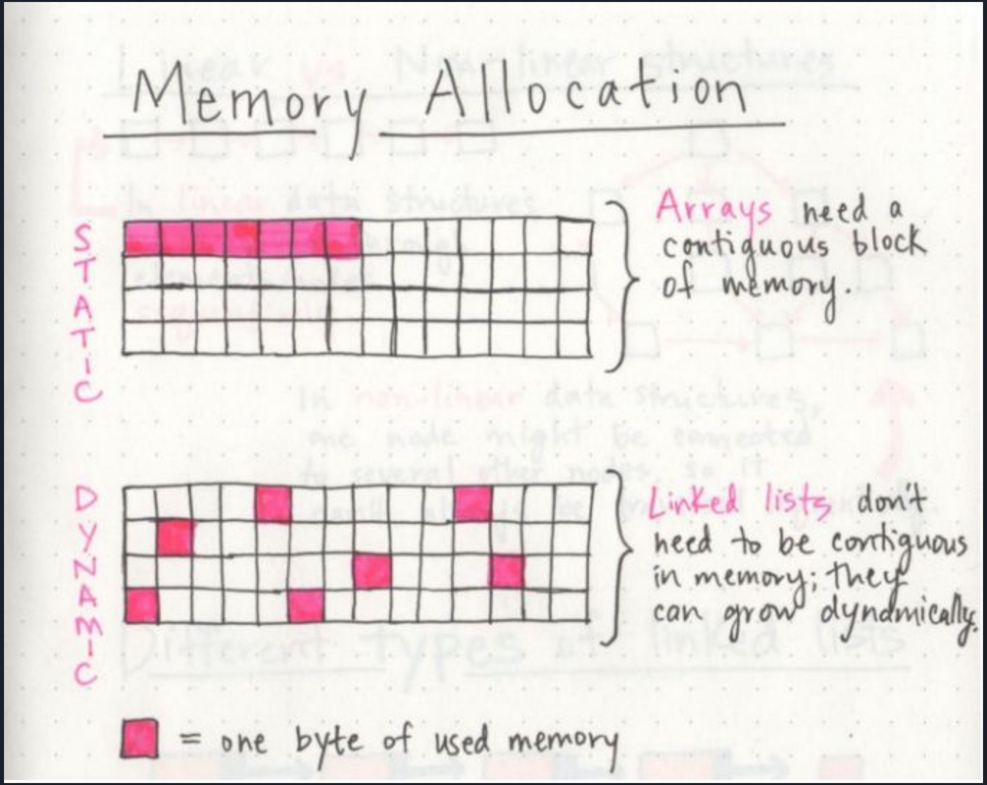
# What's a Linear Data Structure anyways?


Linear vs. Non-linear structures

In linear data structures, we traverse through elements/nodes sequentially.

In non-linear data structures, one node might be connected to several other nodes, so it can't always be traversed sequentially.

## Linked List versus Array


Memory Allocation
STATIC
Arrays need a contiguous block of memory.
DYNAMIC
Linked lists don't need to be contiguous in memory; they can grow dynamically.
🟥 = one byte of used memory

## Linked Lists Require Different Thinking

Linked Lists are NOT index based.

Linked Lists are POINTER based.

How can we access an element using a Linked List as opposed to an array?

Think differently.

We cannot use bracket notation for a Linked List since it is not index based.

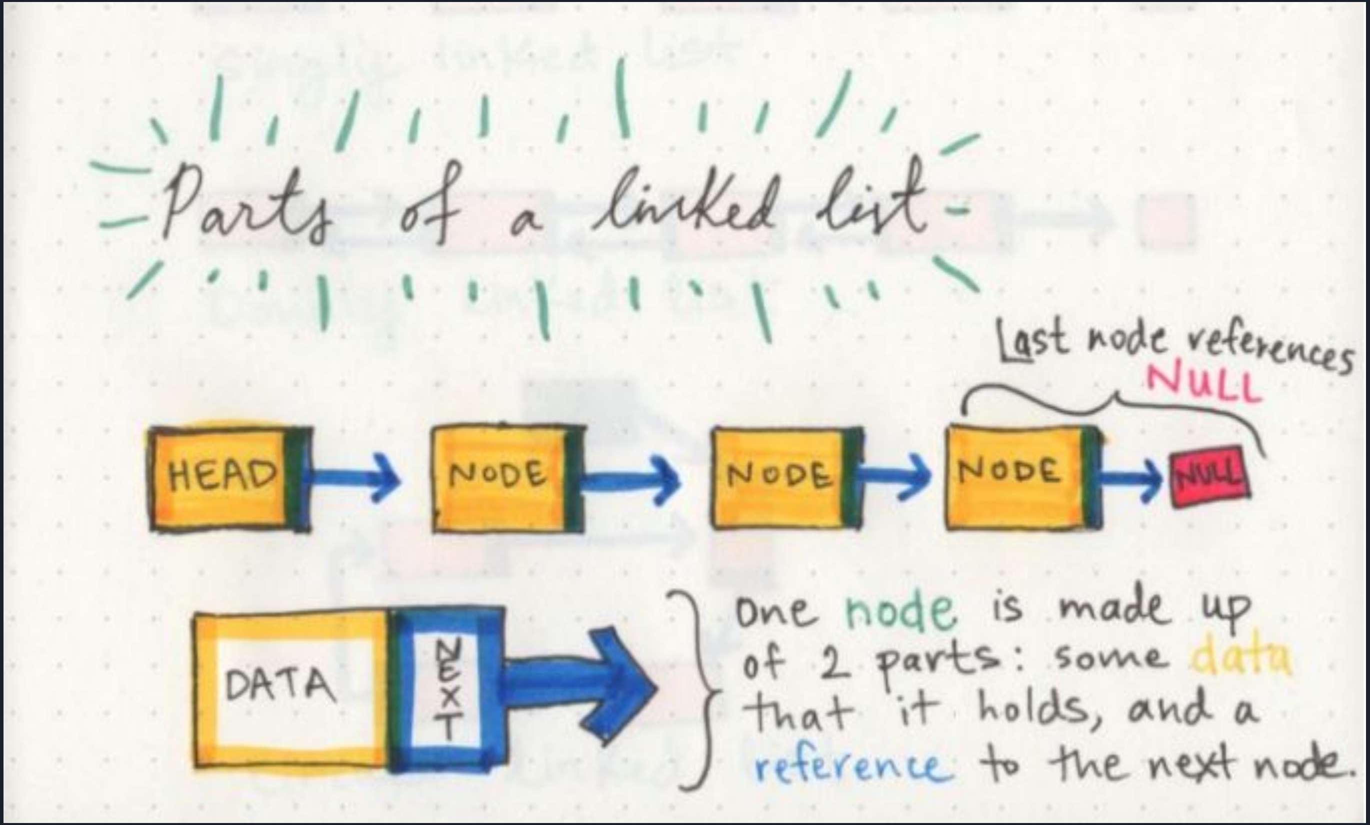
## Linked List Node

How many people here have been asked to “**Reverse a Linked List**” during an interview? Or on LeetCode?

The Linked List Node is the answer!!!!!

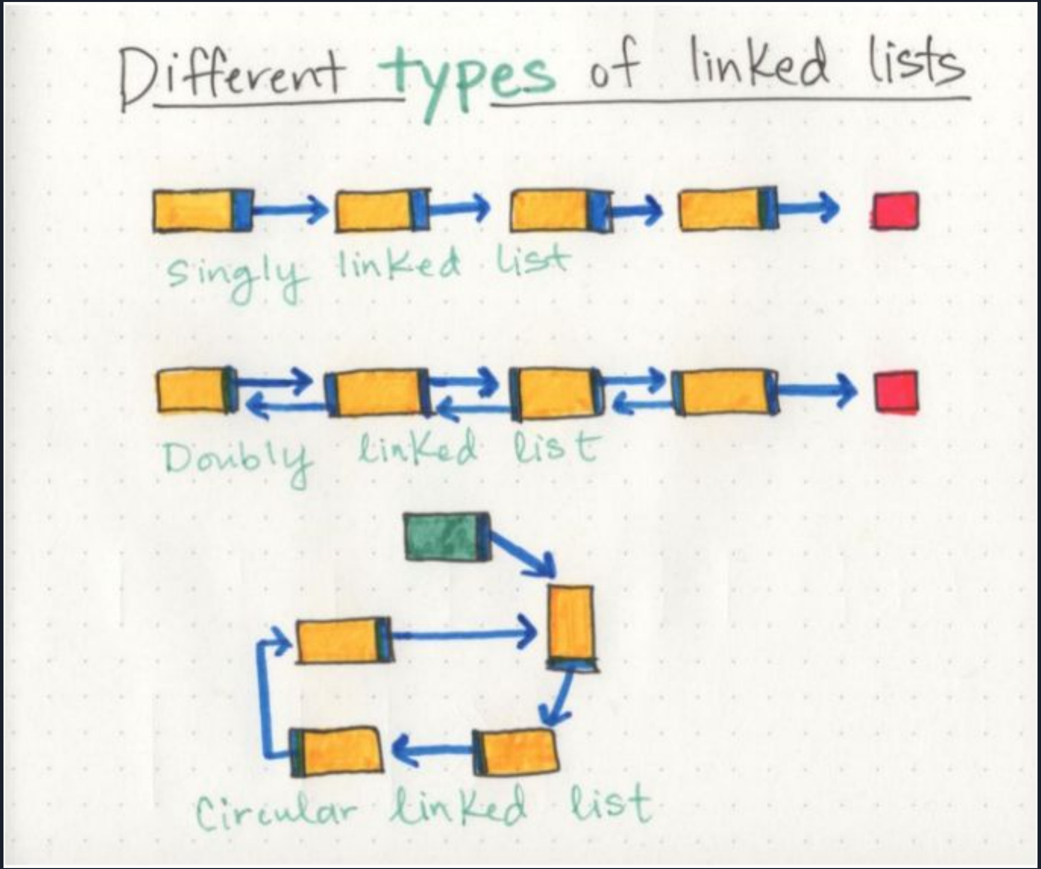
Let’s dissect a Linked List Node.

# Linked List Node



One node is made up of 2 parts: some data that it holds, and a reference to the next node.

# Different Types of Linked Lists



## Visualizations Playground

Let’s head over to visualgo to visualize our data structures in action before we start to code any of them.

[https://visualgo.net/en/list](https://visualgo.net/en/list)

## References

https://medium.com/basecs/whats-a-linked-list-anyway-part-1-d8b7e6508b9d

https://dev.to/b0nbon1/understanding-big-o-notation-with-javascript-25mc