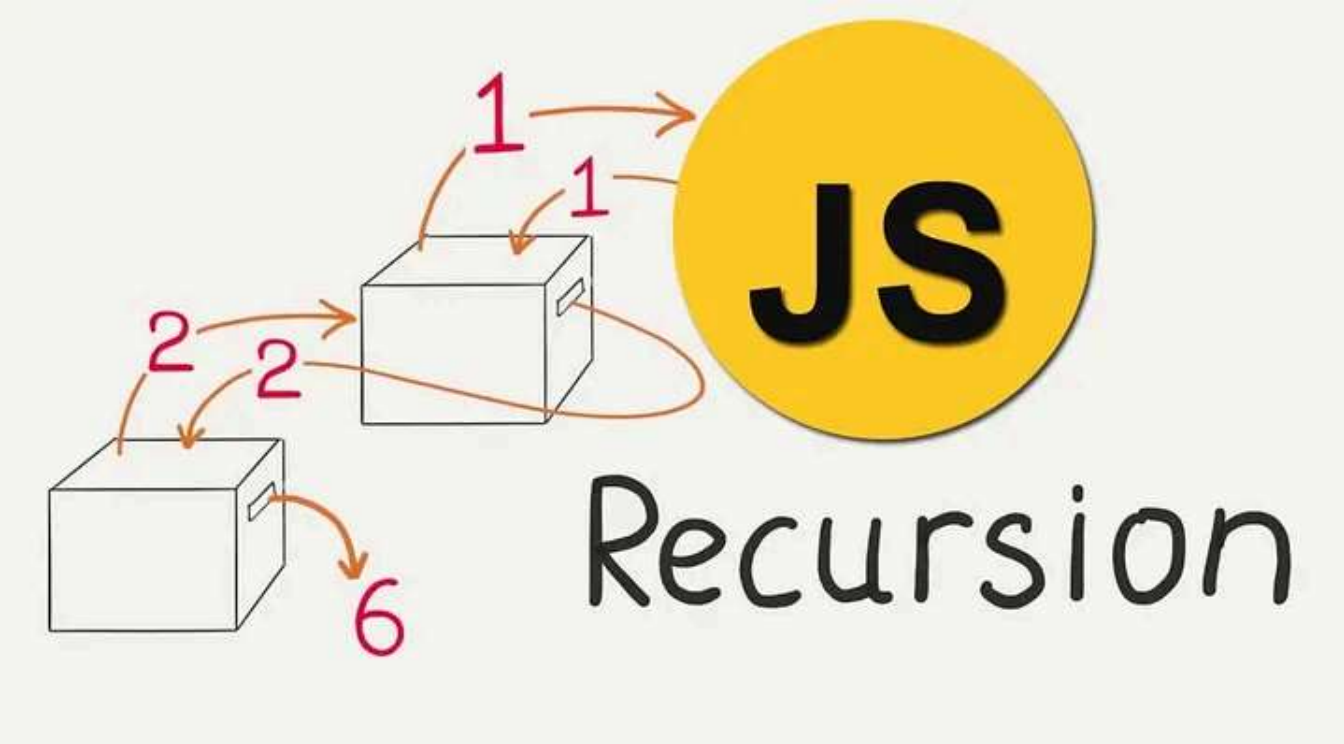
Recursion in JavaScript

Ayush Verma · Follow
5 min read · Feb 5, 2021



What is Recursion?

A **process** (a function in our case) that **calls itself.**

Why do we need to know Recursion?

It's EVERYWHERE!

- Methods using recursion internally — JSON.parse/JSON.stringify, document.getElementById
- DOM traversal algorithms and Object traversal
- A cleaner alternative to iteration.

> ***Call Stack —*** *First let's talk about functions. In almost all program languages there is a built data structure that manages what happens when functions are invoked. Its named as Call Stack in JavaScript.*
>
> *It's a* ***stack*** *data structure. Any time a function is invoked it is placed (****pushed****) on the top of the call stack. When JavaScript sees the* ***return*** *keyword or when the function ends, the complier will remove(****pop****).*
>
> *We are used to functions being pushed on the call stack and popped off when they are done. When we write recursive functions, we keep pushing new functions (in fact the same function)onto the call stack!*

How recursive functions work?

Two essential parts of any recursive functions — **Base case** and **different input.** Invoke the **same** function with a different input until you reach your base case — the condition where the recursion ends.

Examples:

1. **Countdown —** print numbers to the console from whatever number we pass till
    1.

Without recursion:

```
function countDown(num){
    for(var i = num; i > 0; i--) {
       console.log(i);
    }
    console.log("All done!");
}

countDown(5);
```

With recursion:

```
function countDown(num){
    if (num <= 0) {
       console.log("All done!");
       return;
    }
    console.log(num);
    num--;
    countDown(num);
}

countDown(3);

//print 3
//countDown(2)
```

```
//print 2
//countDown(1)
//print 1
//countDown(0) - base case
//print "All done"
```

2. **SumRange —** print sum to the console from whatever number we pass till 1

```
function sumRange(num){
    if (num === 1) return 1;
    return num + sumRange(num-1);
}
sumRange(3);

//return 3 + sumRange(2)
//           return  2 + sumRange(1)
//                       return 1 - base case
// 3 + 2 + 1 = 6
```

3. **Factorial**-print multiplication to the console from whatever number we pass till 1.

Without recursion:

```
function factorial(num){
    let total = 1;
    for(var i = num; i > 0; i--) {
         total *= i;
    }
    return total;
}

factorial(5); //120
```

With recursion:

```
function factorial(num){
    if(num === 1) return 1;
    return num * factorial(num-1);
}

factorial(3); //6
```

Common Recursion Pitfalls

- No base case
- Forgetting to return or returning the wrong thing!

- Maximum call stack size exceeded — stack overflow!

Helper Method Recursion:

A design pattern that's commonly used with recursion.

```
function outer(input){
    var outerScopedVariable = [];
    function helper(helperInput){
       //modify the outerScopedVariable
        helper(helperInput--)
    }
    helper(input)
    return outerScopedVariable;
}

//Two functions - outer(main) and helper (recursive)
//Commonly done when we need to compile an array or list of data.
```

Example —

Collect all the odd values in an array.

```
function collectOddValues(arr){
    let result = [];

    function helper(helperInput){
        if(helperInput.length === 0){
            return;
        }
        if(helperInput[0] % 2 !== 0){
            result.push(helperInput[0]);
        }
        helper(helperInput.slice(1))
    }
    helper(arr)

 return result;
}

collectOddValues([1,2,2,4,4,5,6,7,8]) //(3) [1, 5, 7]
```

Pure Recursion:

```
function collectOddValues(arr){
    let newArr = [];
```

```
    if(arr.length === 0){
        return newArr;
    }
    if(arr[0] % 2 !== 0){
        newArr.push(arr[0]);
    }
    newArr = newArr.concat(collectOddValues(arr.slice(1)));

    return newArr;
}
collectOddValues([1,2,3,4,5]) //(3) [1, 3, 5]

//[1].concat(collectOddValues([2,3,4,5]));
//         [].concat(collectOddValues([3,4,5]));
//                  [3].concat(collectOddValues([4,5]));
//                    [].concat(collectOddValues([5]));
//                      [5].concat(collectOddValues([]));
//                          []
//[1,3,5]
```

Pure Recursion Tips

- For arrays, use methods like **slice**, the **spread** operator, and **concat** that makes copies of arrays so we do not mutate them.
- Remember strings are immutable, so we will need to use methods like **slice**, **substr**, or **substring** to make copies of strings.
- To make copies of object use **Object.assign**, or the **spread** operator.

Recursion examples:

1. **power** — Write a function called power which accepts a base and an exponent. The function should return the power of the base to the exponent. This function should mimic the functionality of **Math.pow()** — do not worry about negative bases and exponents.

```
function power(base, exponent){
    if(exponent === 0) return 1;
    return base * power(base,exponent-1)
}
power(2,0) // 1
power(2,2) // 4
power(2,4) // 16
```

2. **productOfArray —** Write a function called productOfArray which takes in an array of numbers and returns the product of them all.

```
function productOfArray(arr){
    if(arr.length === 0) return 1;
    return arr[0] * productOfArray(arr.slice(1))
}

productOfArray([1,2,3]) // 6
productOfArray([1,2,3,10]) // 60
```

3. **Fibonacci —** Write a recursive function called fib which accepts a number and returns the nth number in the Fibonacci sequence. Recall that the Fibonacci sequence is the sequence of whole numbers 1, 1, 2, 3, 5, 8, … which starts with 1 and 1, and where every number thereafter is equal to the sum of the previous two numbers.

```
function fib(n){
    if (n <= 2) return 1;
    return fib(n-1) + fib(n-2);
}

fib(4) //3
fib(6) //8
fib(10) //55

//n = 4
//fib(3) + fib(2)
//[fib(2)+ fib(1)] + 1
//1 + 1 + 1
//3

//n = 6
//fib(5) + fib(4)
//[fib(4)+ fib(3)] + [fib(3)+ fib(2)]
//[fib(3)+ fib(2) + fib(2)+ fib(1)] + [fib(2)+ fib(1) + 1]

//[fib(2) + fib(1)+ fib(2) + fib(2)+ fib(1)] + [fib(2)+ fib(1) + 1]
//[1 + 1 + 1 + 1 + 1] + [1 + 1 + 1]
//8
```

4. **reverse —** Write a recursive function called reverse which accepts a string and returns a new string in reverse.

```
function reverse(str){
    if(str.length === 1) return str[0];
    return str[str.length - 1] + reverse(str.slice(0, str.length-1))
}

// reverse('awesome') // 'emosewa'
// reverse('rithmschool') // 'loohcsmhtir'
```

5. **flatten —** Write a recursive function called flatten which accepts an array of arrays and returns a new array with all values flattened.

Helper method:

```
function flatten(arr){
  let resultArr = [];

  function inner(arr){
    for(let i = 0; i< arr.length; i++){
          if(Array.isArray(arr[i])){
              inner(arr[i]);
          }
          else{
              resultArr.push(arr[i])
          }
      }
  }
  inner(arr);
  return resultArr;
}

flatten([1, 2, [3, 4, [5, [6, 7, [[[8]]]]]]]) //[1, 2, 3, 4, 5, 6,
7, 8]
```

Pure recursion:

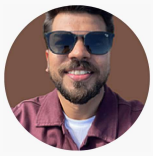
```
function flatten(arr){
  let resultArr = [];
  for(let i = 0; i < arr.length; i++){
      if(Array.isArray(arr[i])){
          resultArr = resultArr.concat(flatten(arr[i]))
      }
      else{
          resultArr.push(arr[i])
      }
  }
  return resultArr;
}

// flatten([1, 2, 3, [4, 5] ]) // [1, 2, 3, 4, 5]
// flatten([1, [2, [3, 4], [[5]]]]) // [1, 2, 3, 4, 5]
```

```
// flatten([[1],[2],[3]]) // [1,2,3]
// flatten([[[[1], [[[2]]], [[[[[[[3]]]]]]]]]) // [1,2,3]
```

Web Development JavaScript Recursion Javascript Development

Functions In Javascript



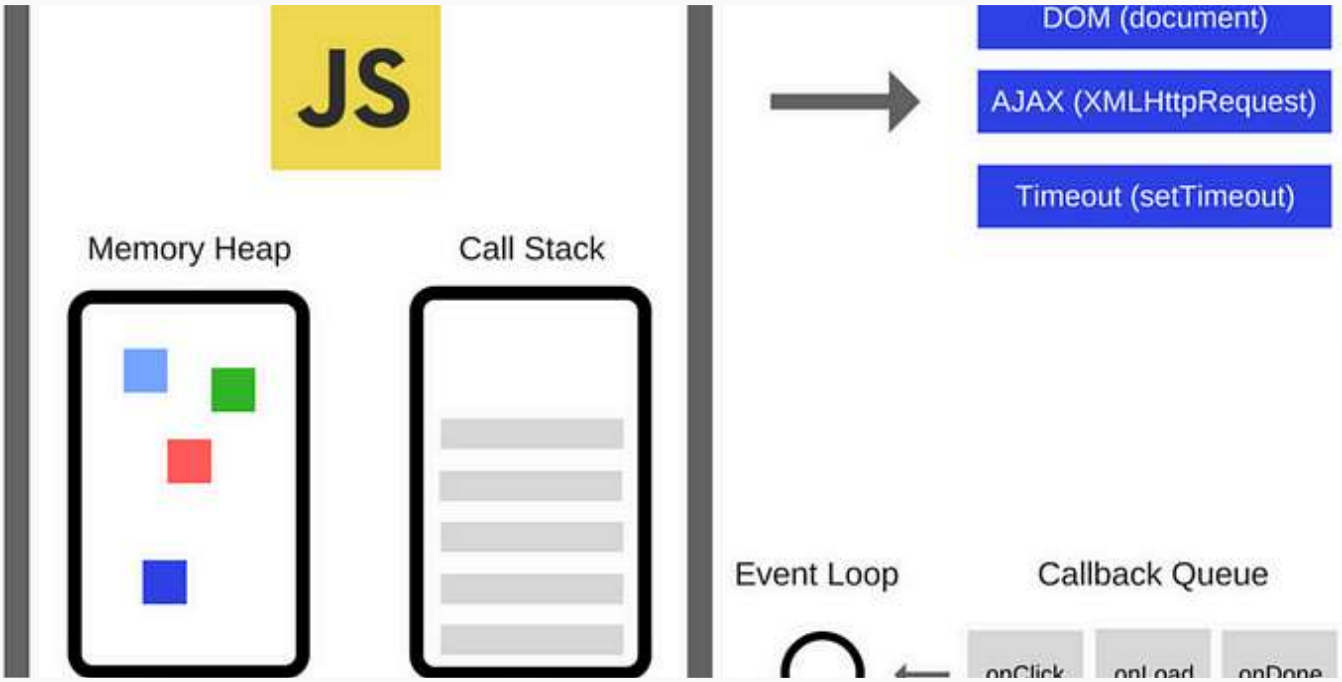
Follow

Written by Ayush Verma

611 Followers

Web developer who loves to code and help others code :)

More from Ayush Verma



Ayush Verma in Towards Dev

The JavaScript Event Loop: Explained

Call Stack, Web APIs, Event Queue, Micro-tasks, and Macro-tasks

11 min read · Apr 14, 2021

1.1K 4

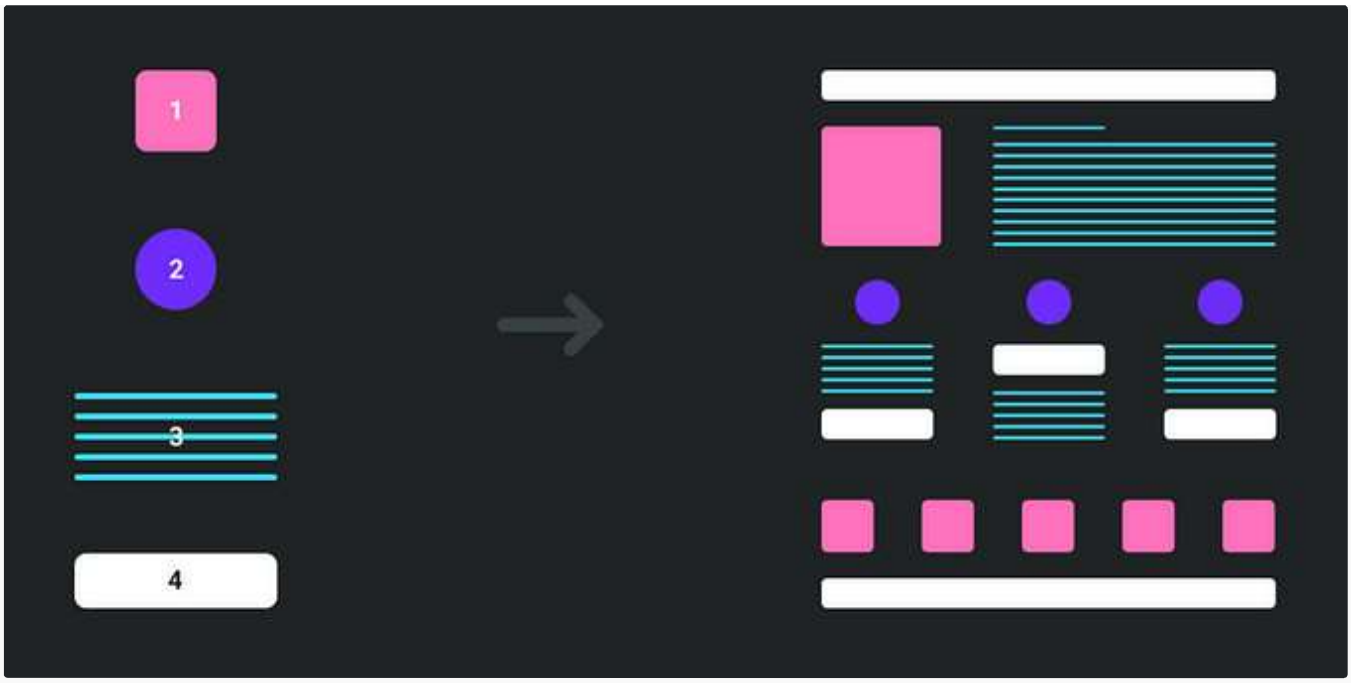
Ayush Verma in JavaScript in Plain English

50 JavaScript Output-related Interview Questions

Interview Q/A with explanations—Strings, Numbers, Boolean, Objects, Arrays, setTimeout, “this” keyword.

16 min read · Apr 14, 2021

397 3



Ayush Verma in JavaScript in Plain English

React: All About Components

Dumb Component, Smart Component, Pure Component, Higher-Order Component, Controlled & Uncontrolled Component

8 min read · Sep 12, 2021

232

Ayush Verma in Code Crunch

JavaScript Array Methods: Explained with Polyfills—forEach, map, filter, reduce & splice

forEach, map, filter, reduce & splice—implementation and polyfills

4 min read · Feb 9, 2021

92 2

See all from Ayush Verma

Recommended from Medium

Satria Suria

Event propagation in JavaScript

What is event propagation?

3 min read · Oct 9, 2023

47

Francesco Saviano

JavaScript Objects: 10 Real-World Exercises

In the dynamic world of web development, JavaScript stands as a cornerstone, powering interactive and sophisticated web applications. At...

21 min read · Jan 1, 2024

5

Lists

General Coding Knowledge

20 stories · 1091 saves

Stories to Help You Grow as a Software Developer

19 stories · 960 saves

Coding & Development

11 stories · 549 saves

Tech & Tools

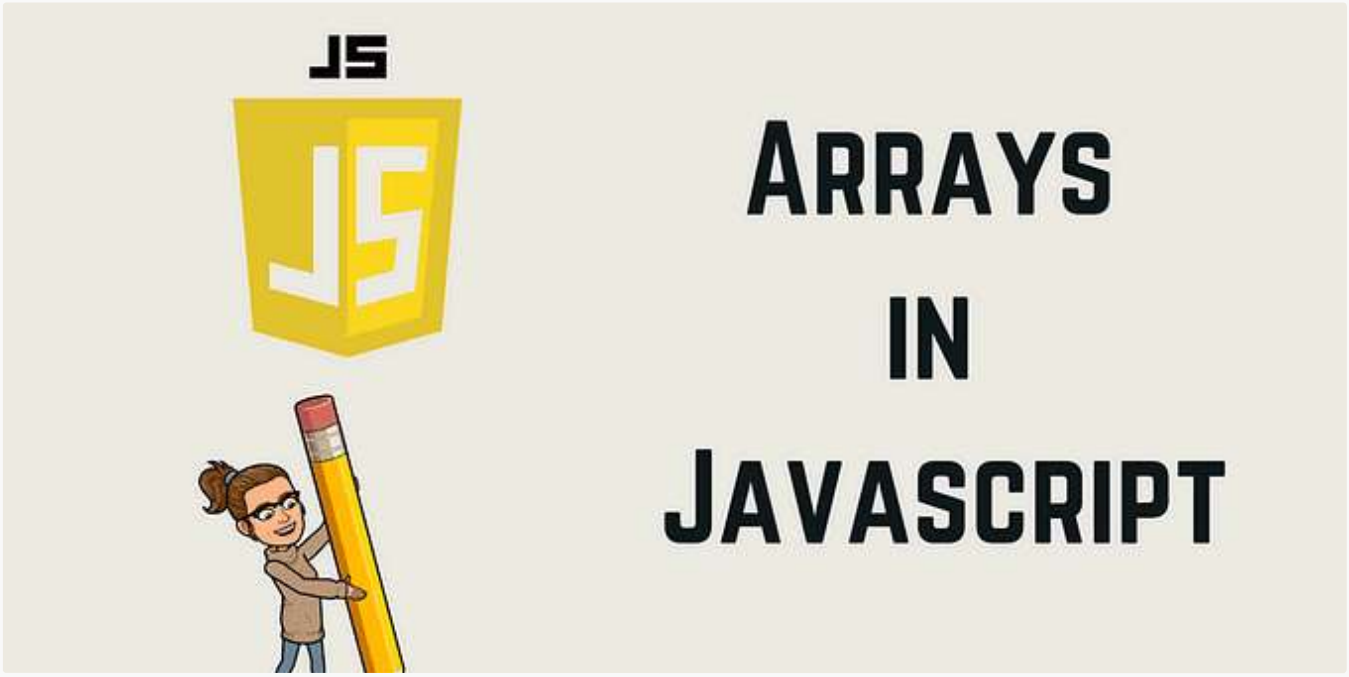
16 stories · 198 saves

devtalib

JavaScript arrays

An array is a special variable that can hold more than one value. It is a collection of elements of the same type, stored in contiguous...

1 min read · Oct 29, 2023



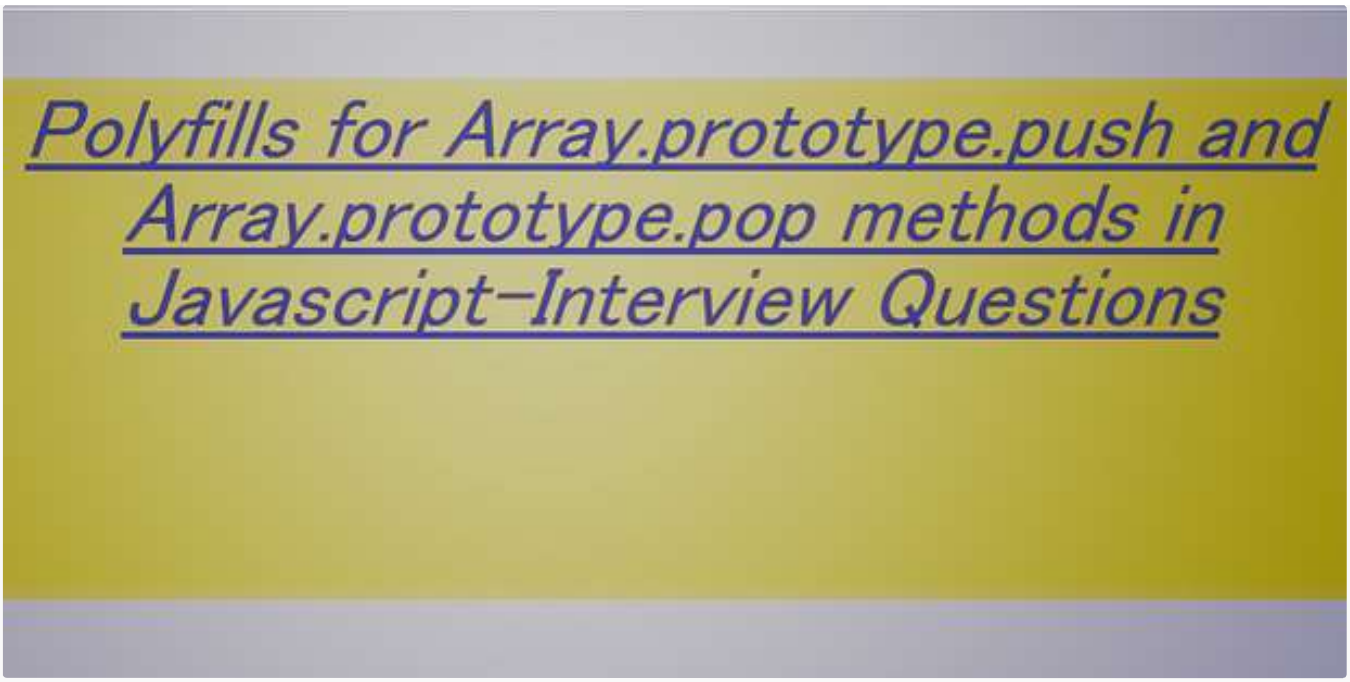
DavitDvalashvili

JavaScript Array Methods: Your Complete Guide

In JavaScript, an array is a special type of object used to store and organize multiple values. Arrays enable you to group values under a...

12 min read · Feb 13, 2024

50



Shaik Abdul Sameer in Dev Genius

Polyfill for Array.prototype.push and Array.prototype.pop methods—Interview Questions

So we all know how to use push and pop methods if we are using an array. But have we ever wondered how the internal built-in methods work...

3 min read · Jan 31, 2024

62

PROFESSOR !!

Diff between map and for-Each

map:

1 min read · Feb 3, 2024

 6  1 

See more recommendations