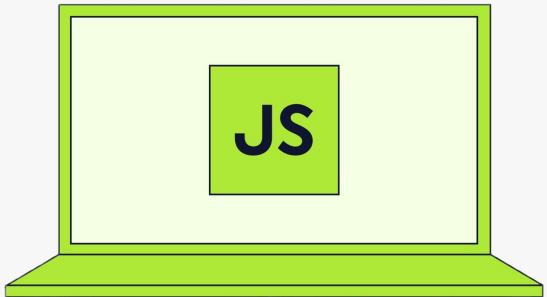




The Complete Javascript Course



@newtonschool

Lecture 5: Introduction to Arrays

-Vishal Sharma



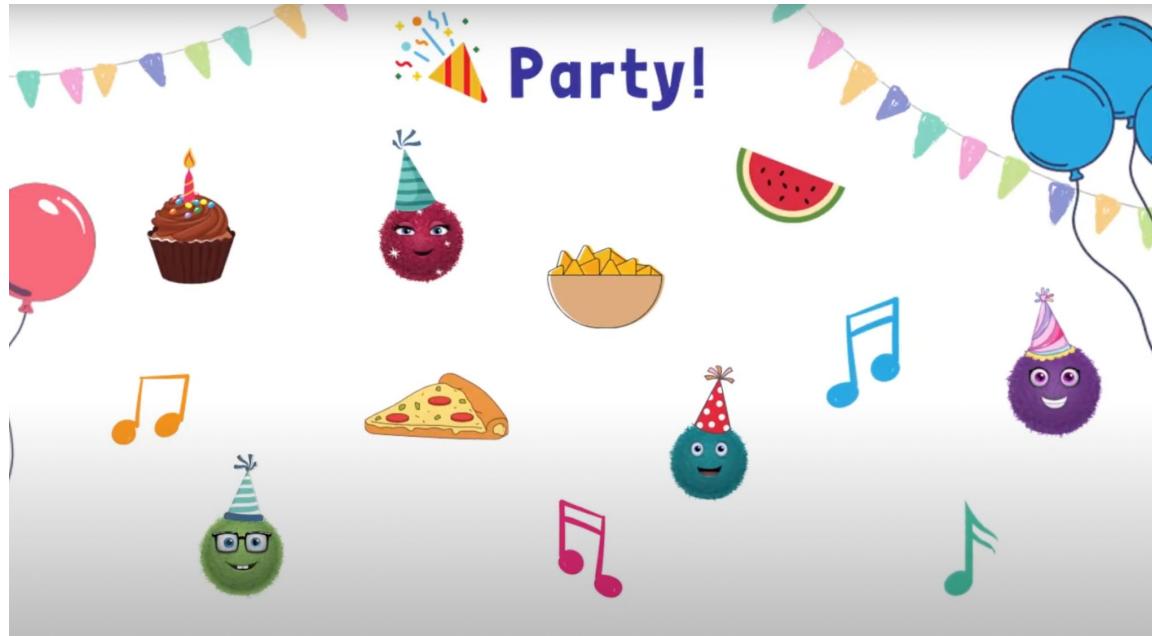
Table of Contents

- What are Arrays?
- Javascript Arrays vs Python Lists
- Common methods:-
 - push
 - pop
 - shift
 - unshift
 - find
 - includes
 - sort
- Bonus Methods: join and split
- Call by Value vs Call by Reference

Arrays

Planning a party

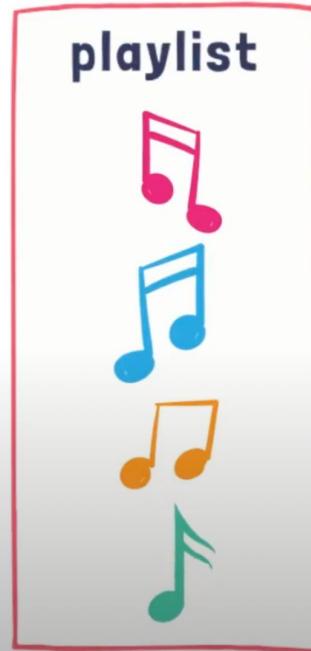
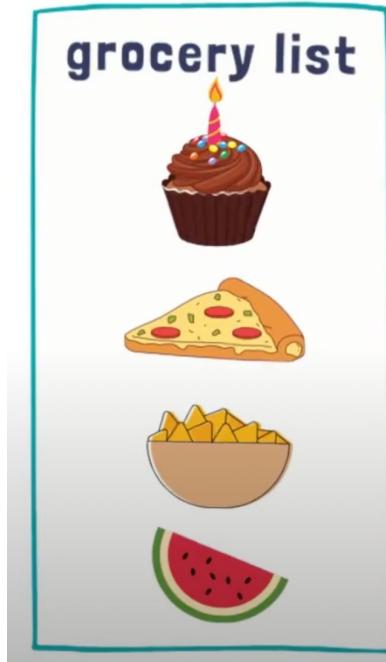
Imagine you are organizing a party, for that you need to purchase lot of items.



You will need, cake, music player, food and decorative items. So you might want to make a list out of it!!

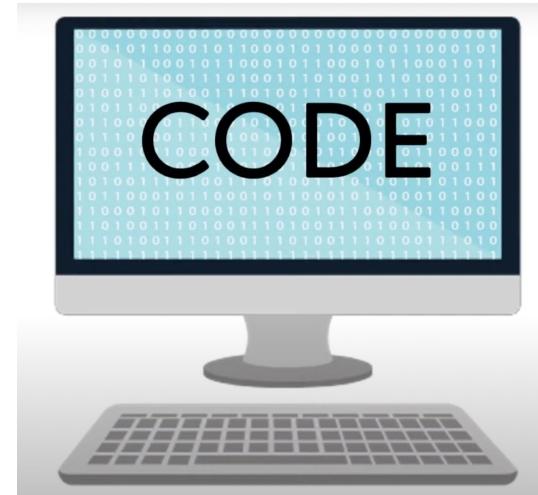
Planning a party: making lists

Let's make lists, separate list for separate group of items:-



Planning a party: making lists

Just like lists help you keep your tasks organized. Lists are very helpful in programming too:-



Array: Lists in javascript

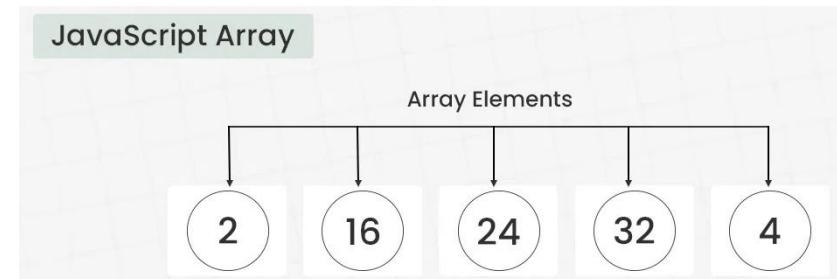
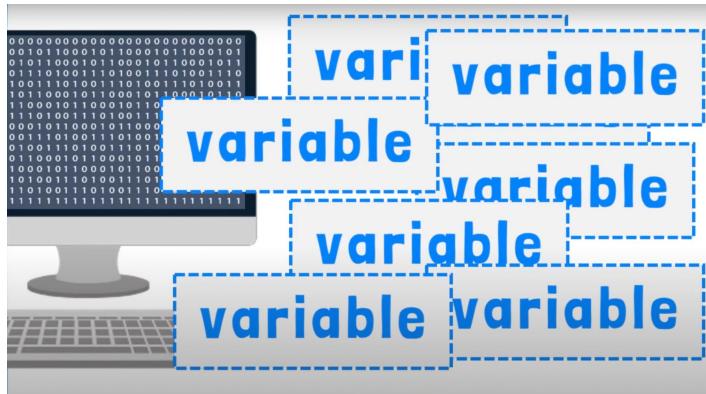
Lists in javascript are called “Array”.

Definition: An array in JavaScript is a special object used to store multiple values in a single variable.



Why do we need array?

Variables store data, but managing many of them is hard. Arrays simplify this by organizing multiple values in one place.

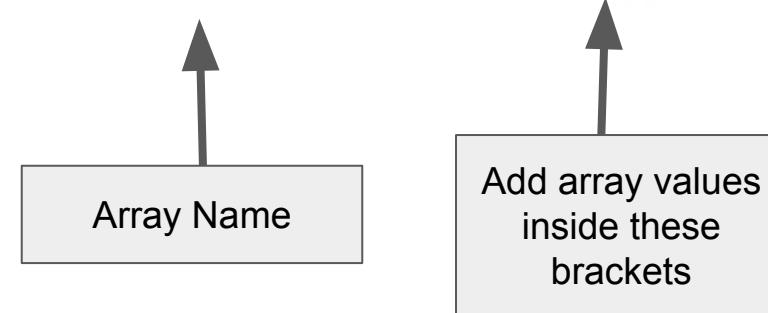


Array: Definition and Syntax

An array in JavaScript is a special object used to store multiple values in a single variable.

Const

MyArray = [] ;



Array: Storing values in arrays

We can declare array as well as store values at the same time.

```
Const MyArray = [ 1, 2, 3, 4, "Hello", true, null ] ;
```



Array Name



You can use mixed data
type in arrays

Array: Accessing the array values

We can access values stored in array by using indexes.

```
1 // Declaring and assigning array
2 const MyArray = [1, 2, 3, 4, "Hello", true , null];
3
4 // Printing array values
5 console.log(MyArray);
6 // Output: [1, 2, 3, 4, "Hello", true , null];
7 console.log(MyArray[0]); // Output: 1
8 console.log(MyArray[1]); // Output: 2
9 console.log(MyArray[4]); // Output: Hello
10 console.log(MyArray[5]); // Output: True
```

If we just type
MyArray in console,
entire array would get
printed

Array Manipulation

Array manipulation

Often, we need to manipulate arrays by adding, removing, updating, or extracting elements. Array methods make this process easy and efficient.



We achieve it using array
methods.

Common Javascript Methods

Here are some commonly used array methods:

push

pop

find

unshift

shift

includes

Add Items

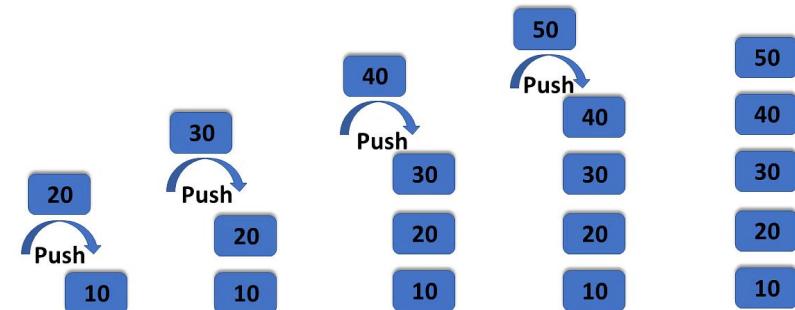
Remove items

Search

Add Items: push

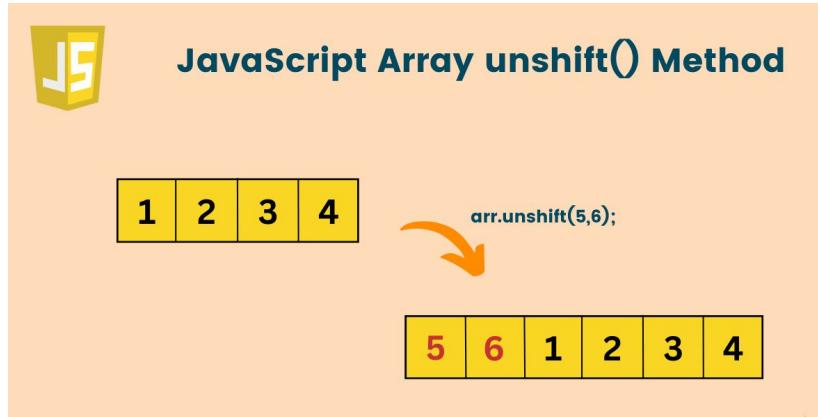
Adds one or more items to the **end** of the array. Initially there was 10, however we could have started with empty array

```
1 // Declaring an empty array
2 let numbers = [10];
3
4 // Adding items one by one
5 numbers.push(20); // [20]
6 numbers.push(30); // [20, 30]
7 numbers.push(40); // [20, 30, 40]
8 numbers.push(50); // [20, 30, 40, 50]
9
10 console.log(numbers); // [20, 30, 40, 50]
11
12 // Adding two items at once
13 numbers.push(60, 70);
14
15 console.log(numbers); // [20, 30, 40, 50, 60, 70]
```



Add Items: unshift

Instead of adding items at the end of the array we can add items to the **beginning** of the array.

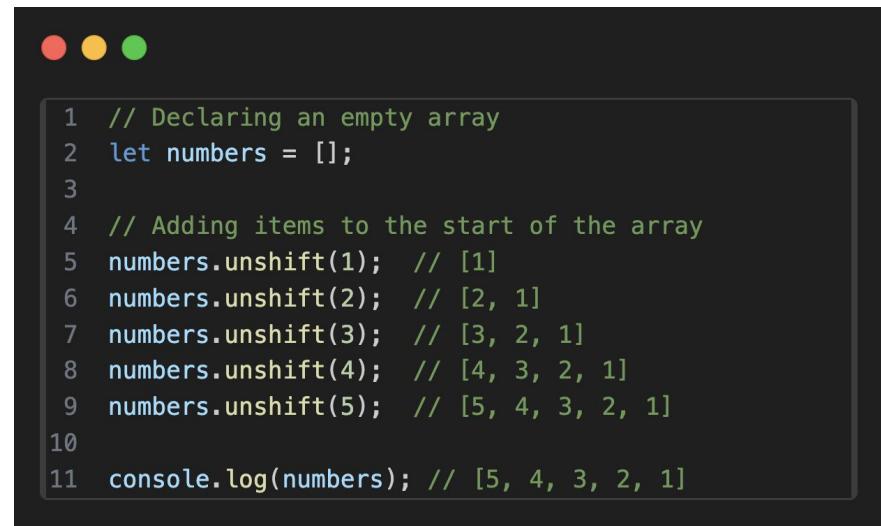


JavaScript Array unshift() Method

1	2	3	4
---	---	---	---

arr.unshift(5,6);

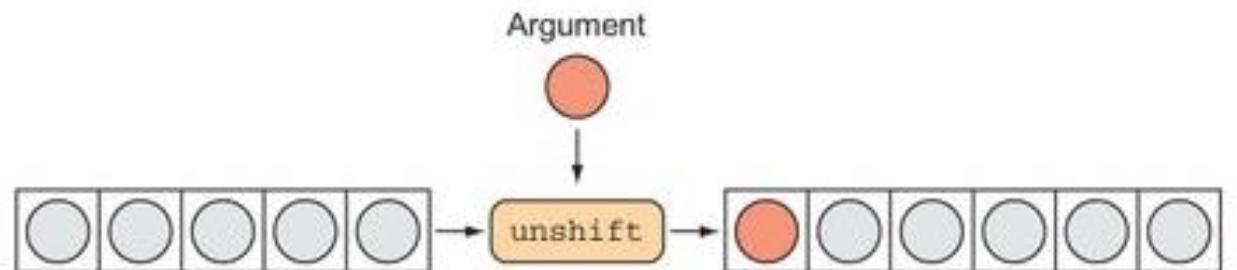
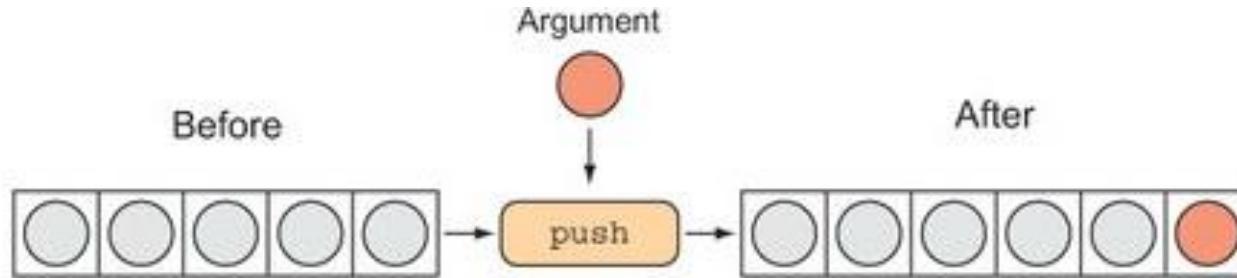
5	6	1	2	3	4
---	---	---	---	---	---



```
1 // Declaring an empty array
2 let numbers = [];
3
4 // Adding items to the start of the array
5 numbers.unshift(1); // [1]
6 numbers.unshift(2); // [2, 1]
7 numbers.unshift(3); // [3, 2, 1]
8 numbers.unshift(4); // [4, 3, 2, 1]
9 numbers.unshift(5); // [5, 4, 3, 2, 1]
10
11 console.log(numbers); // [5, 4, 3, 2, 1]
```

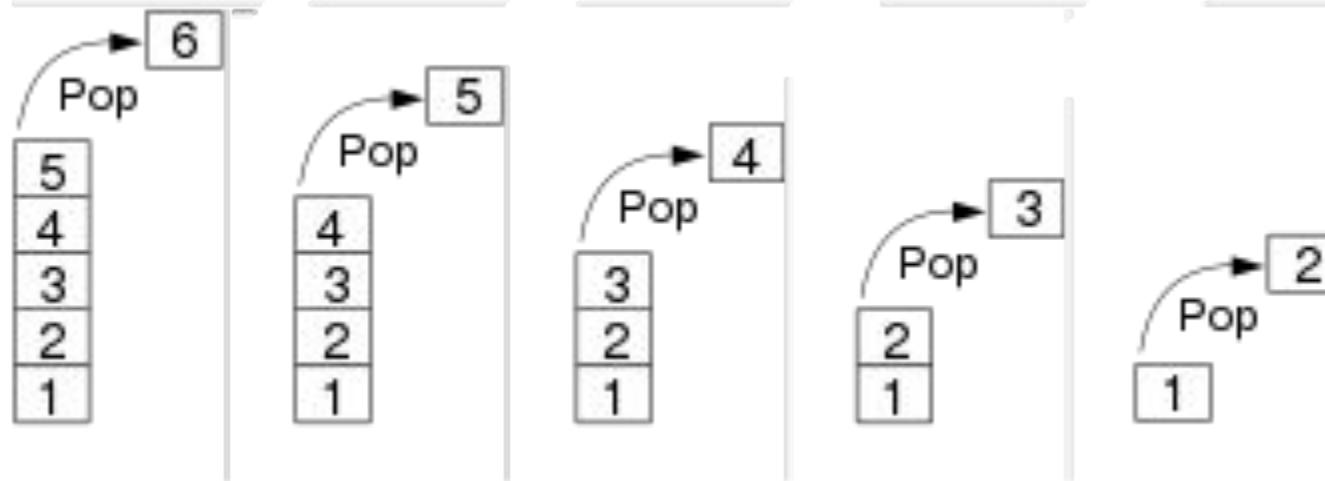
Difference: push and unshift

Let's understand the difference between them two:-



Remove Items: pop

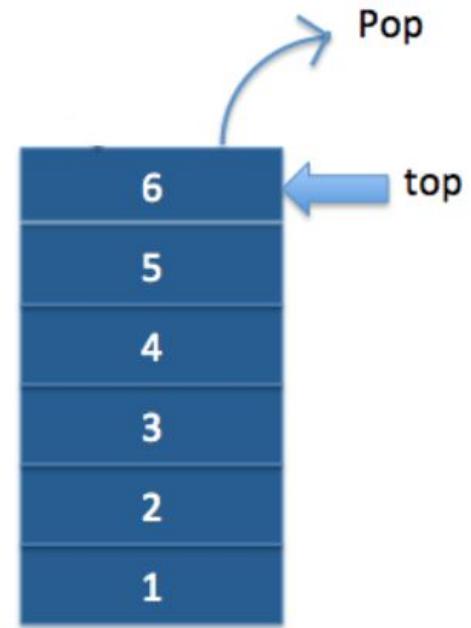
Removes the **last** item from an array.



Remove Items: pop

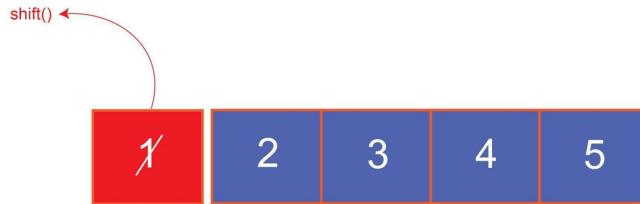
Let's write javascript code for it:-

```
1 // Initial array with 5 items
2 let numbers = [1, 2, 3, 4, 5];
3
4 // Using pop to remove all items
5 numbers.pop(); // [1, 2, 3, 4]
6 numbers.pop(); // [1, 2, 3]
7 numbers.pop(); // [1, 2]
8 numbers.pop(); // [1]
9 numbers.pop(); // []
10
11 console.log(numbers); // []
```



Remove Items: shift

Removes the **first** item from an array.

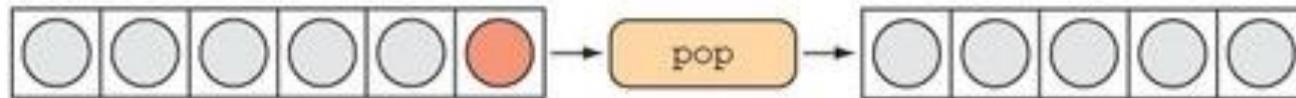


A screenshot of a terminal window on a Mac OS X system, indicated by the red, yellow, and green window control buttons at the top. The terminal displays the following JavaScript code:

```
1 // Initial array with 5 items
2 let numbers = [1, 2, 3, 4, 5];
3
4 // Using shift to remove items
5 numbers.shift(); // [2, 3, 4, 5]
6 numbers.shift(); // [3, 4, 5]
7 numbers.shift(); // [4, 5]
8 numbers.shift(); // [5]
9 numbers.shift(); // []
10
11 console.log(numbers); // []
```

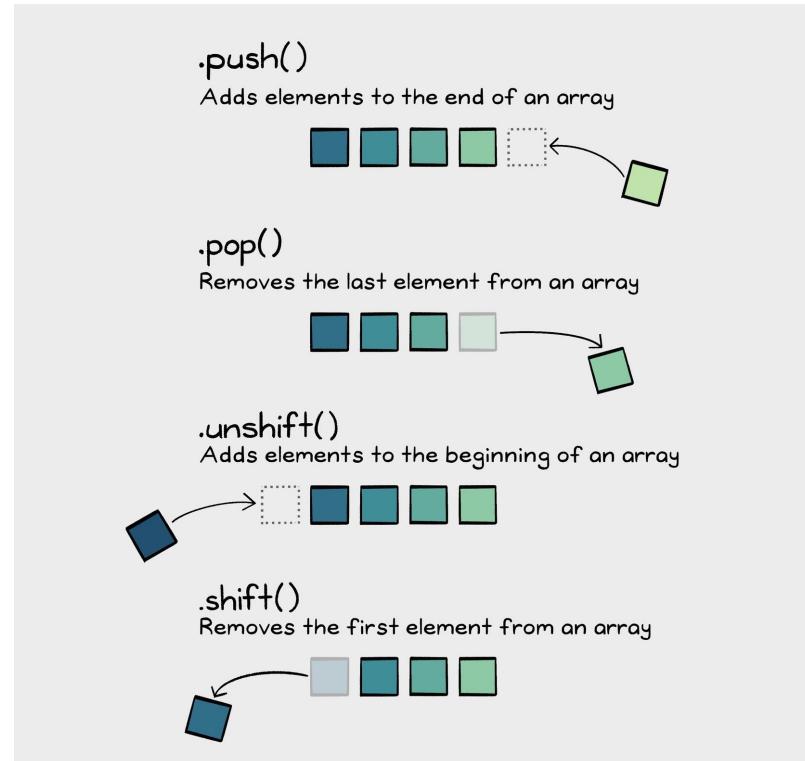
Difference: pop and shift

Let's understand the difference between these two:-



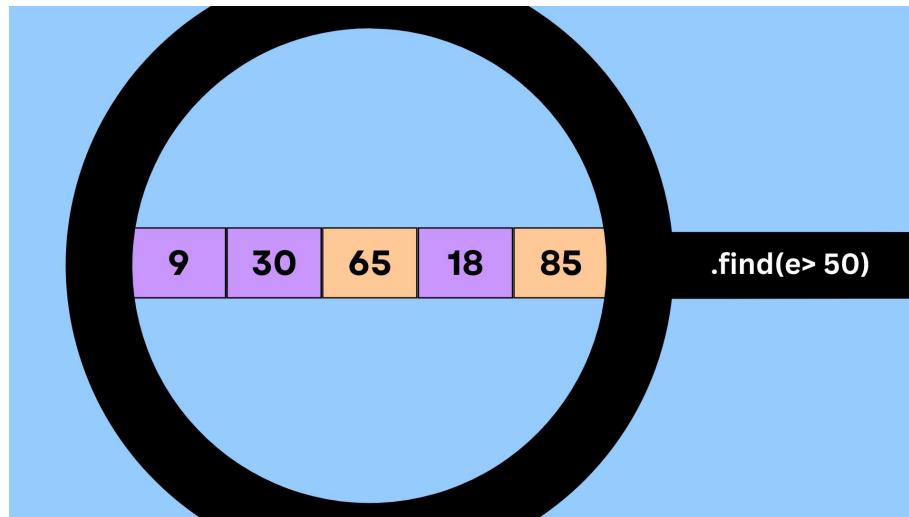
Let's quickly recap what we learned

Here's a quick recap of the array methods we've covered:



Search Methods: `find()`

The `find` method helps you search through an array and find the **first** item that matches a specific condition. Once it finds the item, it stops searching and returns that item.



Since only two values i.e. 65 and 85 is greater than 50, but since 65 is the first occurrence, 65 gets returned.

find(): Syntax

Let's have a look at the syntax of find method.

Syntax:



```
1 numbers.find(number) => {  
2     // code ←  
3     // return true/false ←  
4 }
```

find method logic, you can decide it as per as the problem statement

Based on method logic we decide it match is found or not, if found then we return true

find() example 1

You are given an array of integers and your task is to find the first occurrence of a number which is more than 10. Let's use find() method for this purpose:-

```
1 // Define the array of numbers
2 const numbers = [5, 8, 12, 3, 7];
3
4 // `find` method to get the first number greater than 10
5 const greaterThanTen = numbers.find(num => {
6     // Check if the current number is greater than 10
7     const greater = num > 10;
8     return greater;
9 });
10
11 // Log the result to the console
12 console.log(greaterThanTen); // Output: 12
```

Since 12 is the first number which is more than 10, output is 12.

find() example 2

Given an array having data of different people find out the person with exact age of 30 yrs. Let's use an arrow function to shorten the code this time:-



```
1 // Array of objects with name and age
2 let people = [
3   { name: "John", age: 25 },
4   { name: "Alice", age: 30 },
5   { name: "Bob", age: 22 }
6 ];
7
8 // find the first person who is 30 years old
9 let person = people.find(p => p.age === 30);
10
11 // Printing the result
12 console.log(person);
13 // Output: { name: "Alice", age: 30 }
```

We searched the array and printed the information of the person whose age is exactly 30.

Search Methods: includes()

The `includes` method checks if a certain value exists in an array. It returns `true` if the value is found, and `false` if it's not.

```
months.includes("Apr");
```



A diagram illustrating the `includes` method. At the top, the code `months.includes("Apr");` is shown. An orange arrow points from this code down to a dashed-line grid representing an array. The array has five cells, indexed 0 through 4 above them. The cells contain the values "Jan", "Feb", "Mar", "Apr", and "May". The cell containing "Apr" is highlighted with a blue box and labeled "true" with another orange arrow. Below this, the code `const months = [Jan, Feb, Mar, Apr, May];` is shown. A second orange arrow points from the `includes` call in the first code block down to the "Apr" cell in the array grid. A third orange arrow points from the "Apr" cell in the grid down to the `includes` call in the second code block, labeled "false" with an orange arrow.

```
const months = [Jan, Feb, Mar, Apr, May]
```

```
months.includes("apr");
```

includes() example

Given an array of months, check if particular month is present in the array or not.

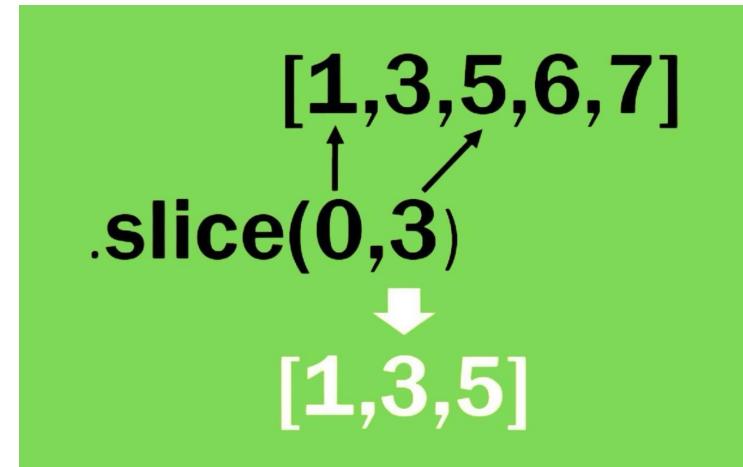
```
1 const months = ["Jan", "Feb", "Mar", "Apr", "May"];
2
3 // Check if the array includes "Apr" and "apr"
4 let includesApr = months.includes("Apr");
5 let includesAprLower = months.includes("apr");
6
7 console.log(includesApr); // Output: true
8 console.log(includesAprLower); // Output: false
```

We haven't stored April in lowercase i.e. "apr" so we got false in second instance.

Copying the array and modifying it directly

Copying the Array: `slice()` method

As the name suggests, the `slice` method cuts out a section of an array and returns a new array containing that section, without modifying the original array.



slice(): Example

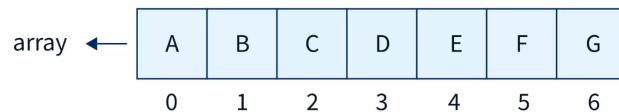
Let's take an initial array [1, 3, 5, 6, 7], and use the slice(0, 3) method to get a portion of it.

```
1 // Initial array
2 let numbers = [1, 3, 5, 6, 7];
3
4 // Using slice from index 0 to 3 (excluding index 3)
5 let slicedArray = numbers.slice(0, 3);
6
7 console.log(slicedArray); // [1, 3, 5]
8 console.log(numbers);
9 // [1, 3, 5, 6, 7] (Original array)
```

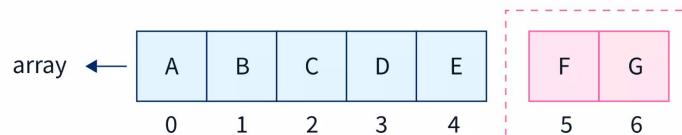
Not the end index
is not included in
the slice!!

Modifying the Array: `splice()` method

The `splice` method changes the contents of an array by removing, replacing, or adding elements at a specified index, modifying the original array.



`array.splice(5, 2)` delete_count is 2, so two elements will be removed starting from index 5



`array.splice(2)` starts removing elements from index 2



`const countries = ["Ghana", "Nigeria", "Rwanda"];`

`countries.splice(1, 0, 'Kenya');`



Removing Items

Adding Items

splice(): Example

`splice(0, 3)` removes 3 elements starting from index 0, modifying the original array to [4, 5].



```
1 // Initial array
2 let numbers = [1, 2, 3, 4, 5];
3
4 // Using splice to remove the first three items
5 numbers.splice(0, 3);
6
7 console.log(numbers); // [4, 5]
```

Here instead of modifying the original array `splice` method modified the original array.

Sort method

Organizing array: `sort()`

The `sort()` method allows us to arrange array elements in a specific order, making data easier to understand and work with.

5	8	2	3	6
---	---	---	---	---

Unsorted Array



2	3	5	6	8
---	---	---	---	---

Sorted Array

Sorting an array organizes data, improves readability and speeds up searches.

sort(): How it works?

By default sort() method sorts in ascending order:-

```
1 // Array declaration and assignment
2 const array = [5, 4, 3, 2, 6];
3
4 // Sorting the array
5 array.sort();
6
7 console.log(array);
8 // Output: [2, 3, 4, 5, 6]
```

We can pass compare function as a callback function to sort() to change its default behaviour.

sort(): How it works?

To sort an array we need to use `sort()` method and pass a compare function to it:-



```
1 // Array declaration and assignment
2 const array = [5, 4, 3, 2, 6];
3
4 // Sorting the array
5 array.sort(compareFunction);
```



```
1 // compare function ascending
2 function compareFunction(a, b){
3     return a - b;
4 }
```



```
1 // compare function descending
2 function compareFunction(a, b){
3     return b - a;
4 }
```

sort(): Example Ascending

Let's sort the array in ascending order:-



```
1 // Sorting the array
2 array.sort(compareFunction);
3
4 // compare function ascending
5 function compareFunction(a, b){
6     return a - b;
7 }
8
9 console.log(array);
10 // Output: [2, 3, 4, 5, 6]
```

If $a - b$ is positive ($a > b$):

- **Action:** b is placed before a because b is smaller.
- **Reason:** The `sort()` function swaps the two elements to ensure the smaller value comes first.

Vice-versa if $a - b$ is negative ($a < b$)

sort(): Example Descending

Let's sort the array in descending order:-



```
1 // Sorting the array
2 array.sort(compareFunction);
3
4 // compare function descending
5 function compareFunction(a, b){
6     return b - a;
7 }
8
9 console.log(array);
10 // Output: [6, 5, 4, 3, 2, 1]
```

If $b - a$ is positive ($b > a$):

- **Action:** b is placed before a because b is larger.
- **Reason:** The sort function keeps b first, ensuring the larger value comes first.

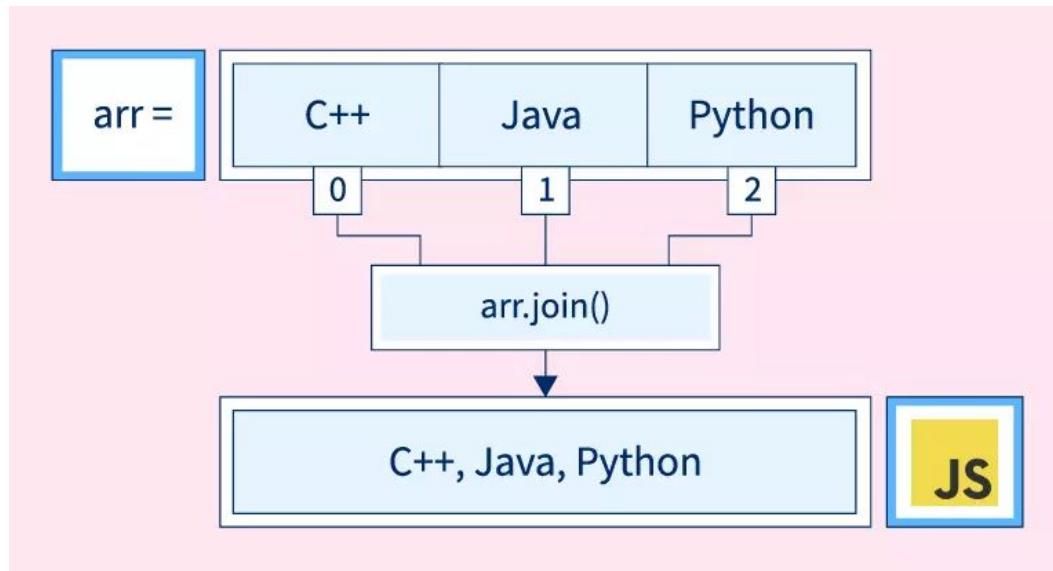
Vice-versa if $b - a$ is negative ($b < a$):

Bonus:

split and join methods

Array to String: join() method

join() method is used to concatenate the array elements into a string.



We can pass separator arguments in join method.

Let's say we did
`arr.join('|')`



C++|Java|Python

join(): Example

Let's understand join method with few examples:-

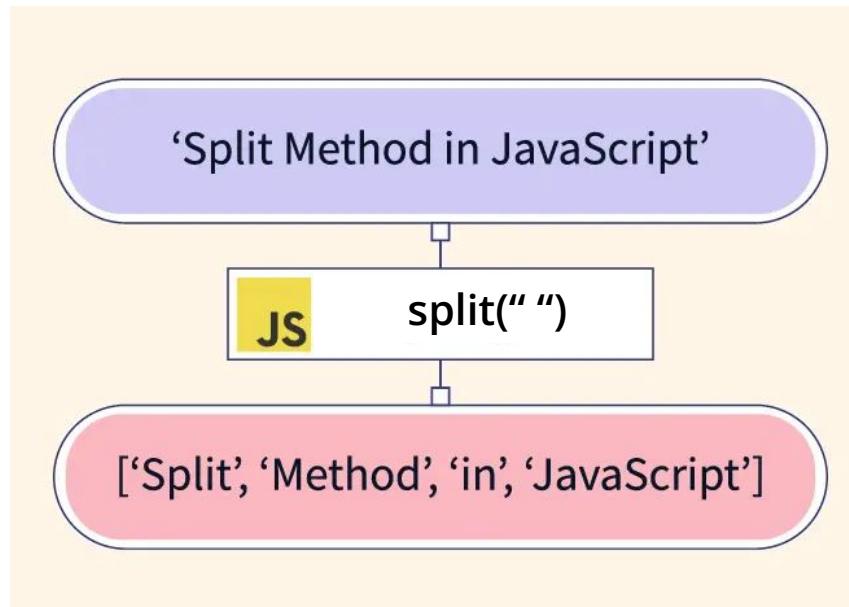


```
1 const languages = ['C++', 'Java', 'Python'];
2
3 // Using the default separator (comma)
4 console.log(languages.join());
5 // Output: "C++,Java,Python"
6
7 // Using a custom separator
8 console.log(languages.join(' | '));
9 // Output: "C++ | Java | Python"
10
11 // Joining without any separator
12 console.log(languages.join(''));
13 // Output: "C++JavaPython"
```

When we don't pass any argument in the join() method we get comma(,) as an separator.

String to Array: split() method

We can reverse the join method i.e. we can create array from string by using split() method:-



We need delimiters inside the split() method. Here in this case we use whitespace:-



split(" ");

split(): Example

Let's understand split method with few examples:-

If we don't pass delimiter it by default returns the whole text



```
1 const text = "C++,Java,Python";
2 const result = text.split();
3 // No separator provided
4
5 console.log(result);
6 // Output: ["C++,Java,Python"]
```

Here we splitted the array using whitespace a delimiter

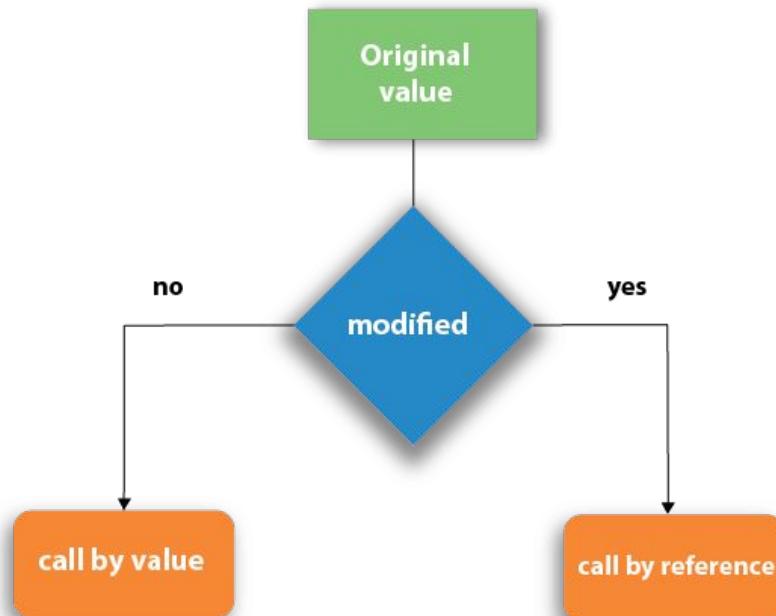


```
1 const sentence = "Split method in javascript";
2 const wordsArray = sentence.split(" ");
3 // Splits the string at each space
4
5 console.log(wordsArray);
6 // Output: ["Split", "method", "in", "javascript"]
```

Call by value vs call by reference

Call by Value vs Call by Reference

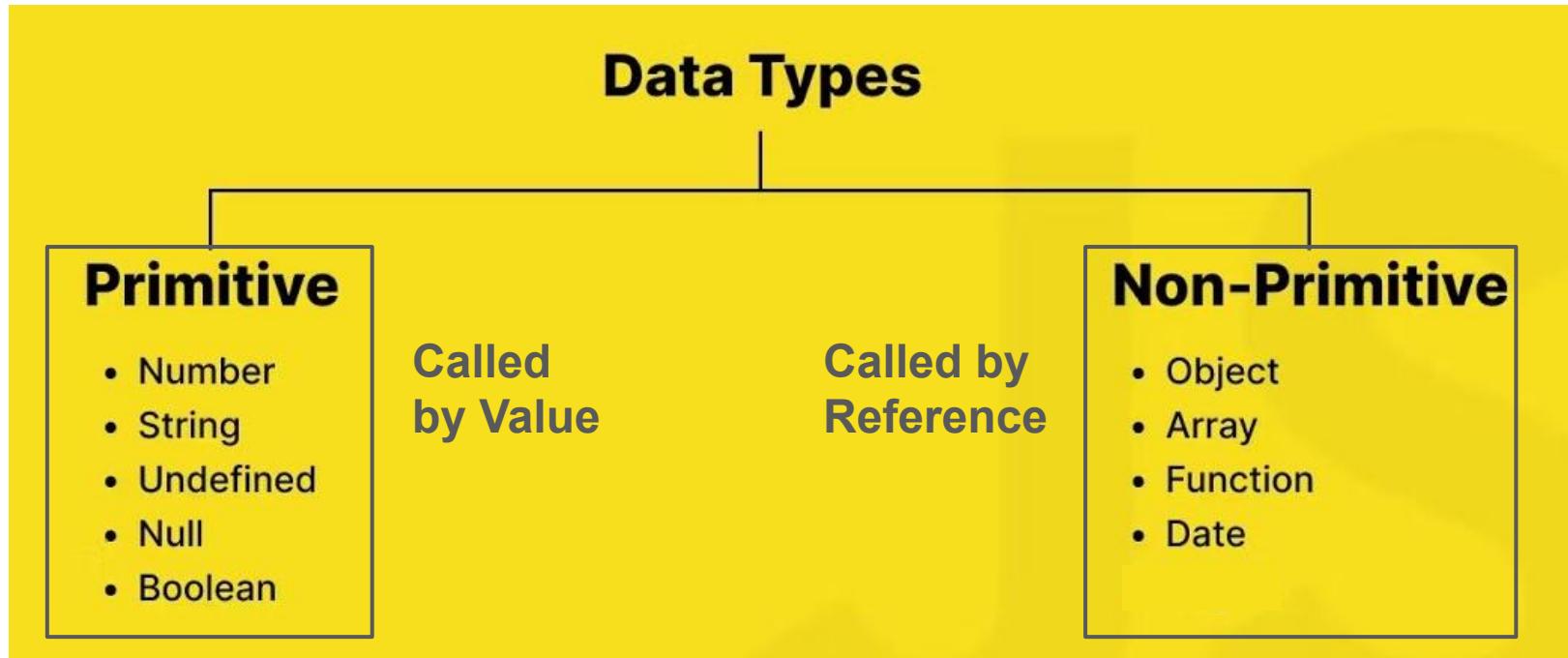
Understanding how JavaScript handles primitives and reference types in functions



Understood?? Don't worry, let's jump into next slides and then come back, to understand.

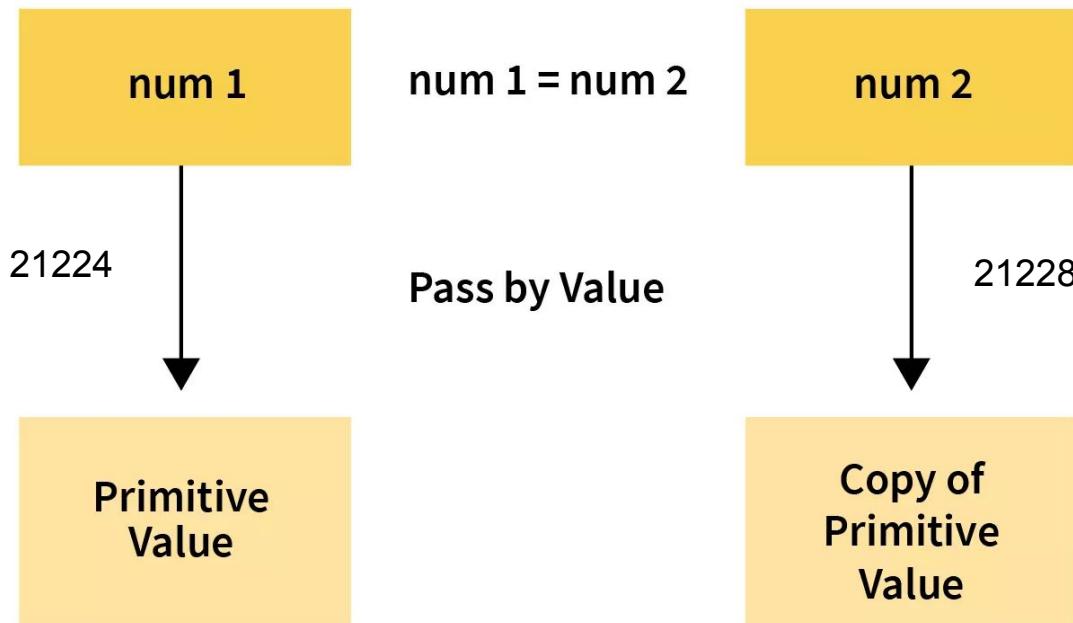
Primitive vs Non-primitive

Primitive data types (like numbers and strings) are passed by value (a copy), while non-primitive data types (like objects) are passed by reference (a link to the original).



Call by Value

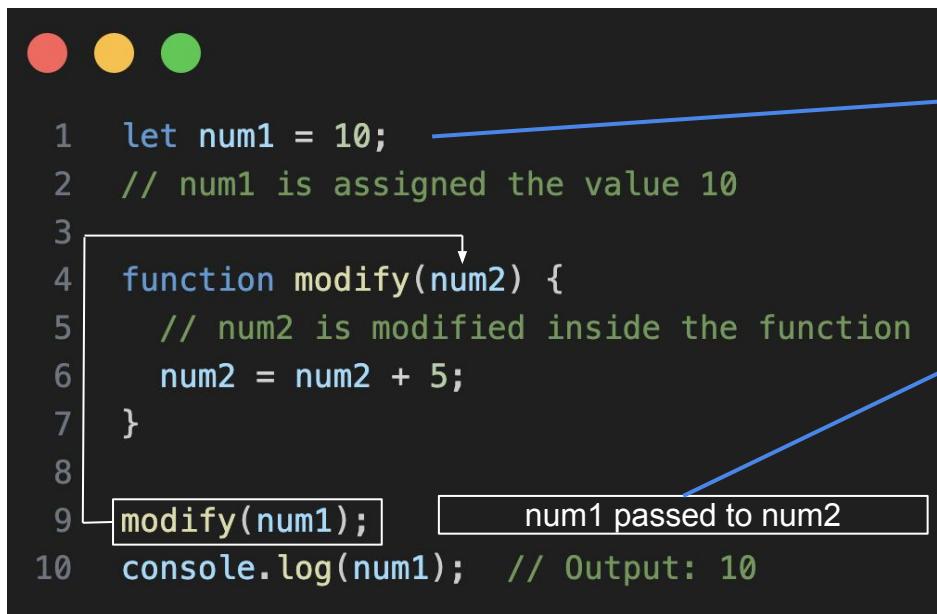
Whenever primitive value is assigned to another variable or passed to a function a new copy of it is generate.



Here 21224 and 21228 are memory locations where num1 and num2 are stored respectively.

Call by Value: example

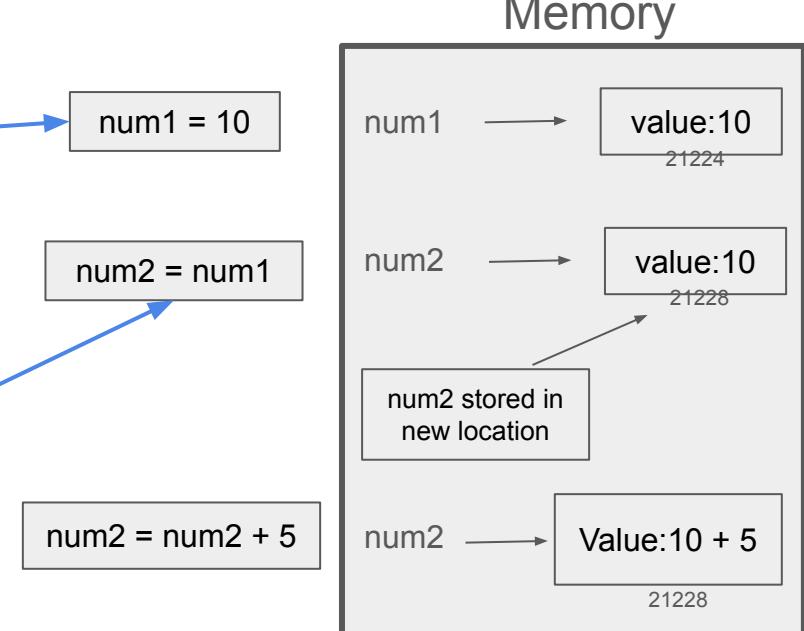
Let's understand how it happens with an example.



```

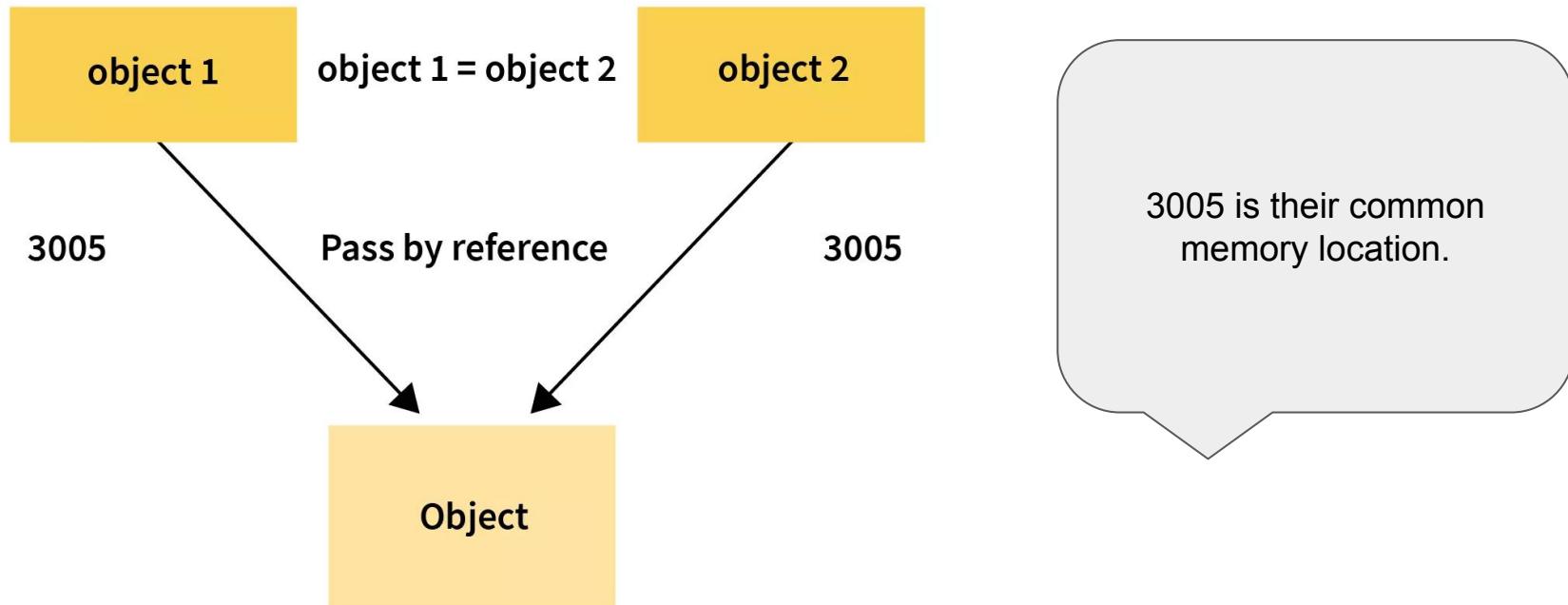
1 let num1 = 10;           num1 = 10
2 // num1 is assigned the value 10

3
4 function modify(num2) {   num2 = num1
5   // num2 is modified inside the function
6   num2 = num2 + 5;
7 }
8
9 modify(num1);           num1 passed to num2
10 console.log(num1); // Output: 10
  
```



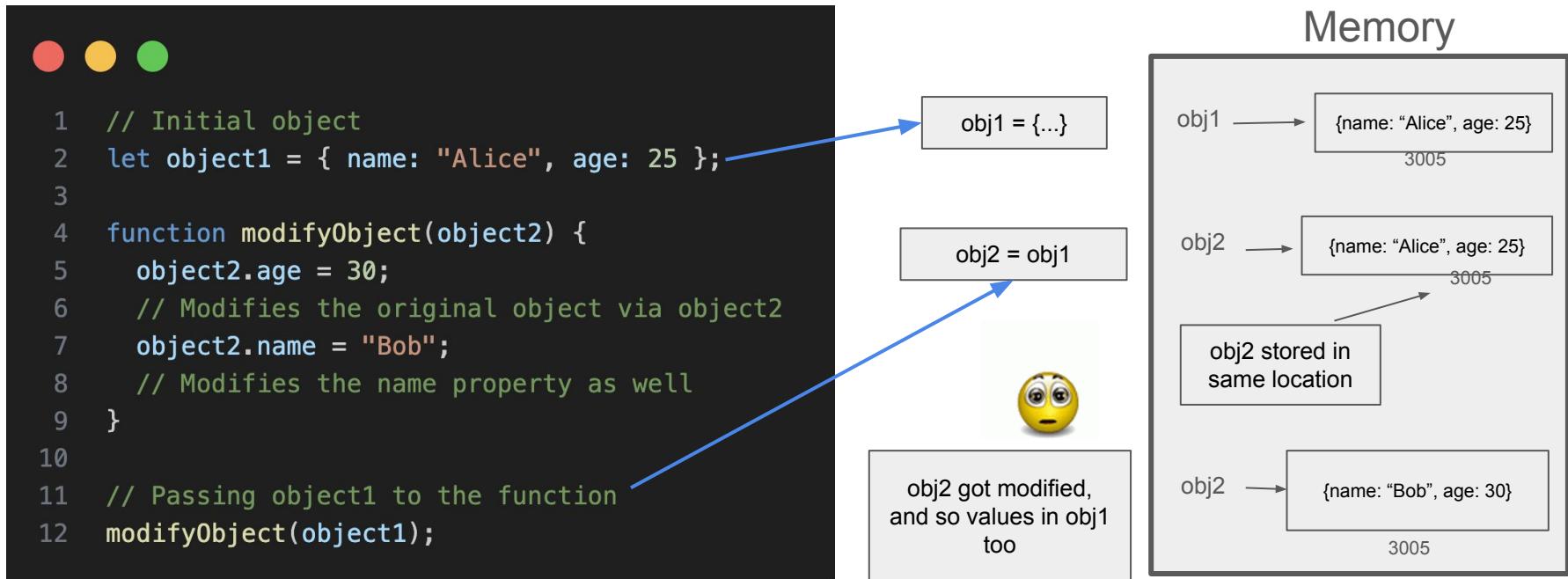
Call by Reference

Whereas in call by reference, the original variable and the function parameter point to the **same memory location**.



Call by Reference: example

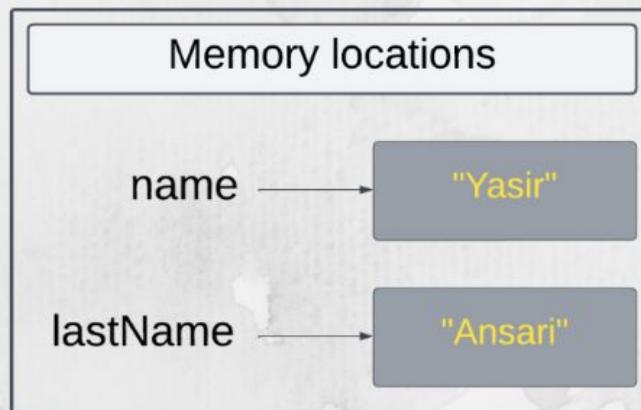
Let's understand how it happens with an example. Here obj1 and obj2 are pointing to same address.



Quick Recap

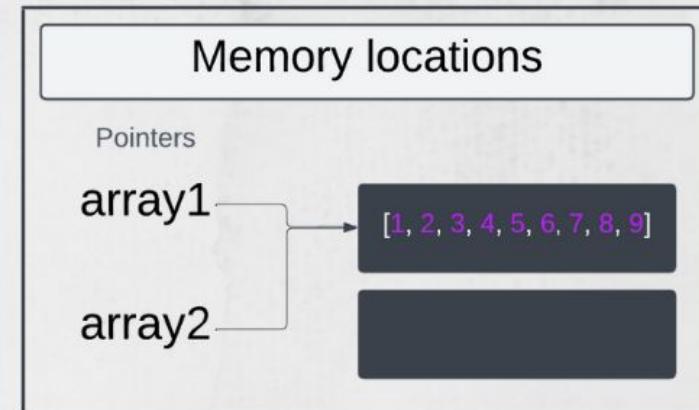
Let's have a quick recap how values are passed based on its type.

Primitive data type



Passed by value.

Reference data type



Passed by reference.



In Class Questions

**Thanks
for
watching!**