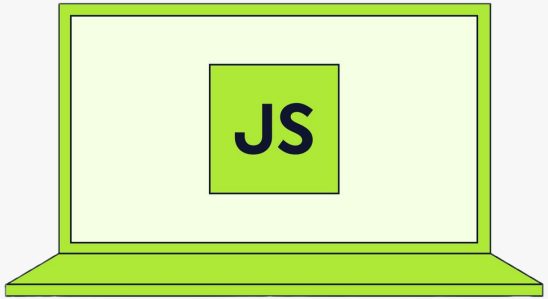
Newton School of Technology

The Complete Javascript Course



@newtonschool

Lecture 5: Introduction to Arrays

-Vishal Sharma

JS

Table of Contents

Newton School of Technology

Newton School of Technology

Arrays

Planning a party

Newton School of Technology

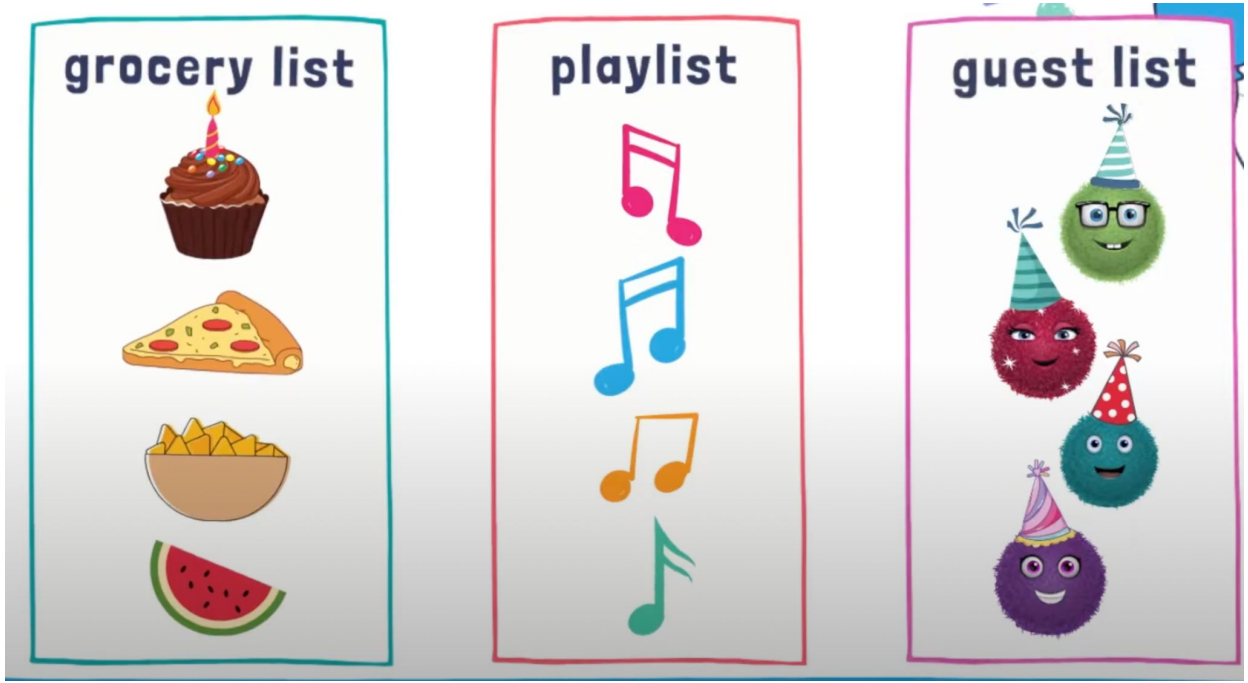
Imagine you are organizing a party, for that you need to purchase lot of items.

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

Newton School of Technology

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

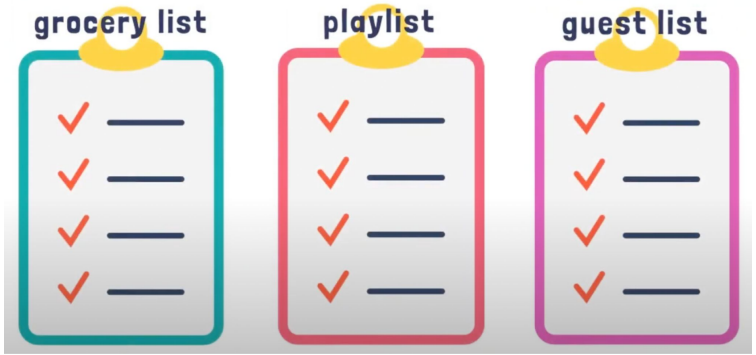


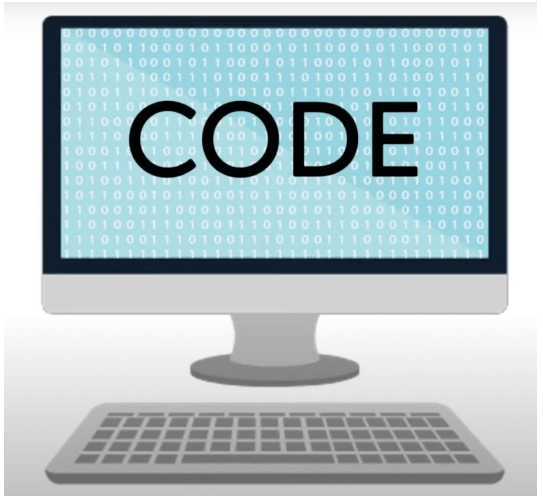
Lists help us to keep our tasks organized!!!

Planning a party: making lists

Newton School of Technology

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





Array: Lists in javascript

Newton School of Technology

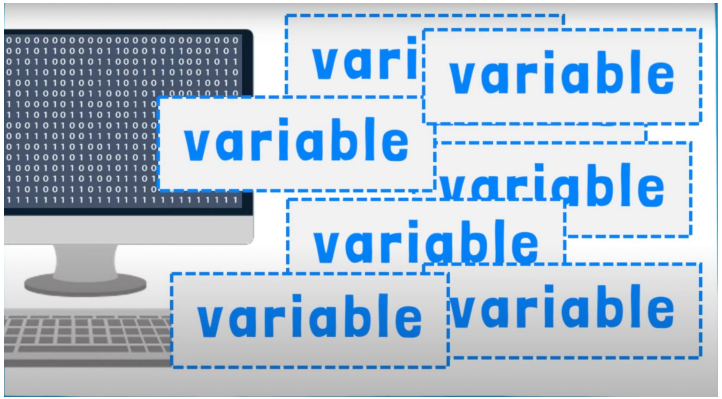
Lists in javascript are called “Array”.

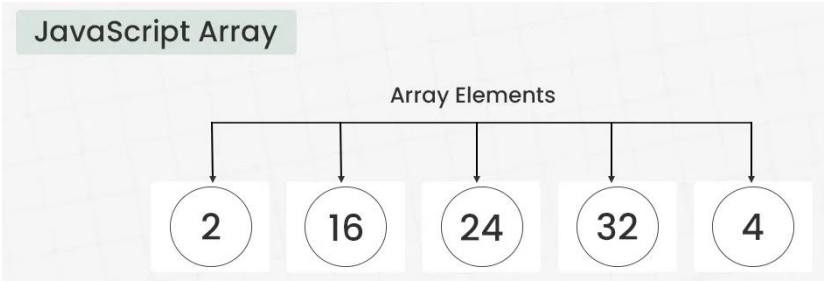
Definition: An array in JavaScript is a special object used to store multiple values in a single variable.

Why do we need array?

Newton School of Technology

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

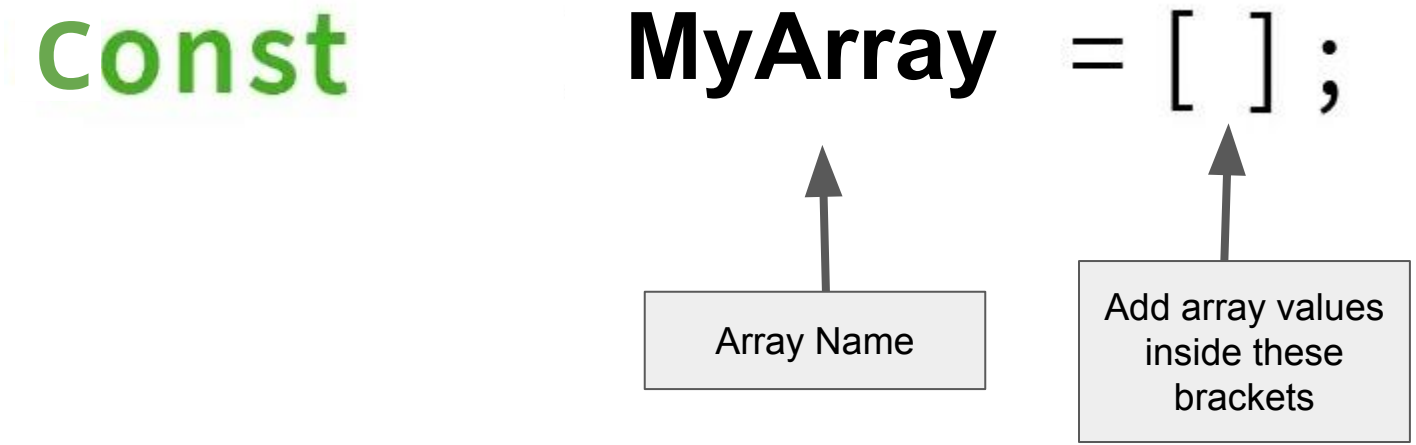




Array: Definition and Syntax

Newton School of Technology

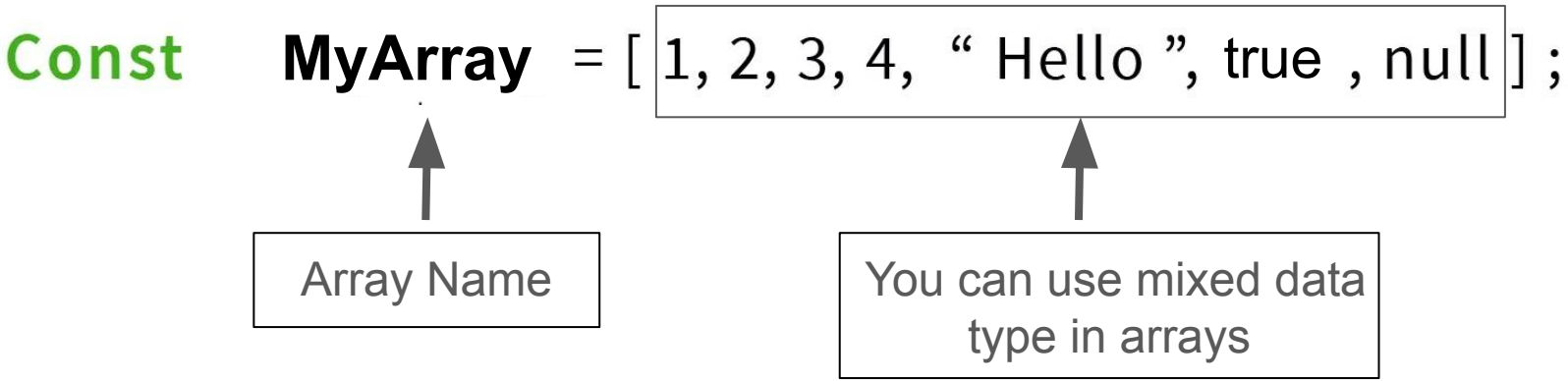
An array in JavaScript is a special object used to store multiple values in a single variable.



Array: Storing values in arrays

Newton School of Technology

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



Array: Accessing the array values

Newton School of Technology

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

Newton School of Technology

Array Manipulation

Array manipulation

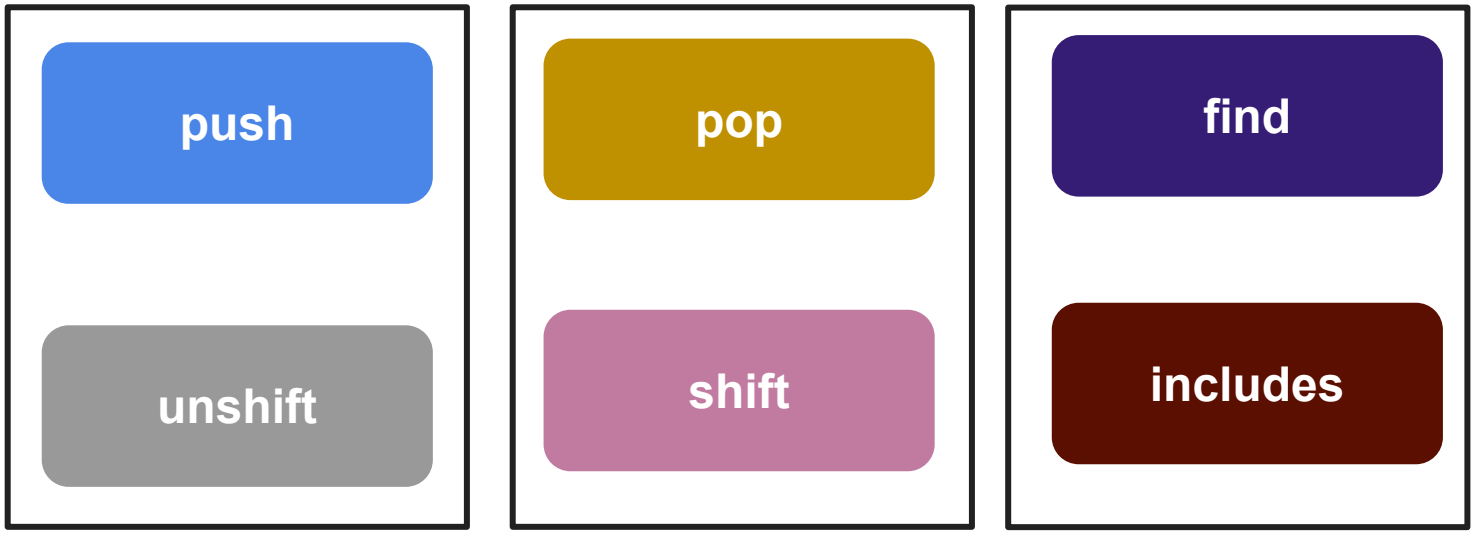
Newton School of Technology

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:

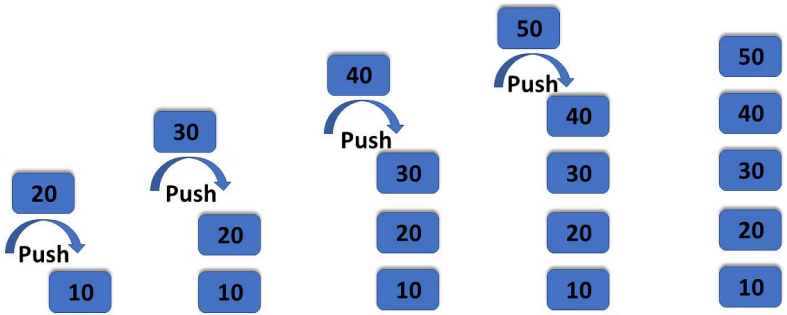


Add Items: push

Newton School of Technology

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

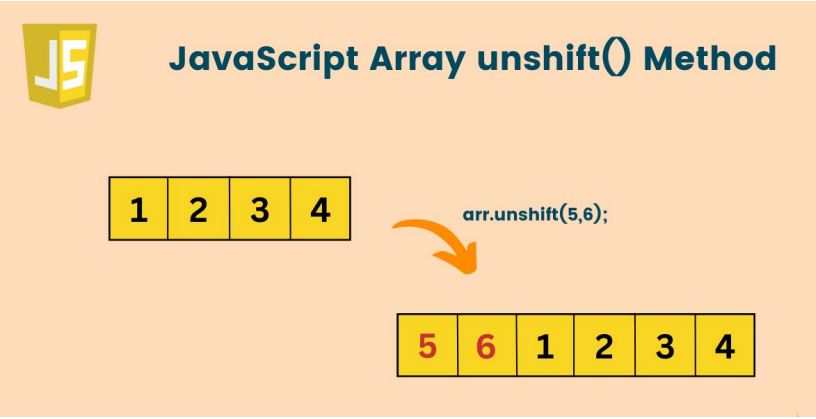
```
// Declaring an empty array
let numbers = [10];

// Adding items one by one
numbers.push(20);  // [20]
numbers.push(30);  // [20, 30]
numbers.push(40);  // [20, 30, 40]
numbers.push(50);  // [20, 30, 40, 50]

console.log(numbers); // [20, 30, 40, 50]

// Adding two items at once
numbers.push(60, 70);

console.log(numbers); // [20, 30, 40, 50, 60, 70]
```



Add Items: unshift

Newton School of Technology

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

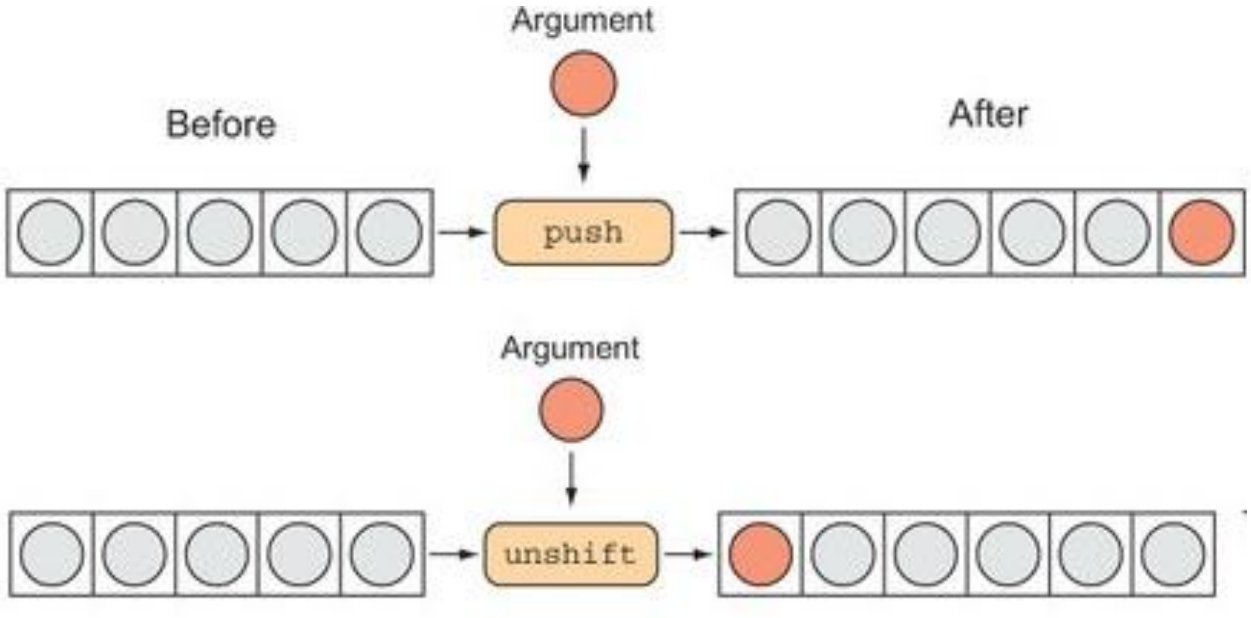



```
// Declaring an empty array
let numbers = [];

// Adding items to the start of the array
numbers.unshift(1);  // [1]
numbers.unshift(2);  // [2, 1]
numbers.unshift(3);  // [3, 2, 1]
numbers.unshift(4);  // [4, 3, 2, 1]
numbers.unshift(5);  // [5, 4, 3, 2, 1]

console.log(numbers); // [5, 4, 3, 2, 1]
```

Difference: push and unshift

Newton School of Technology

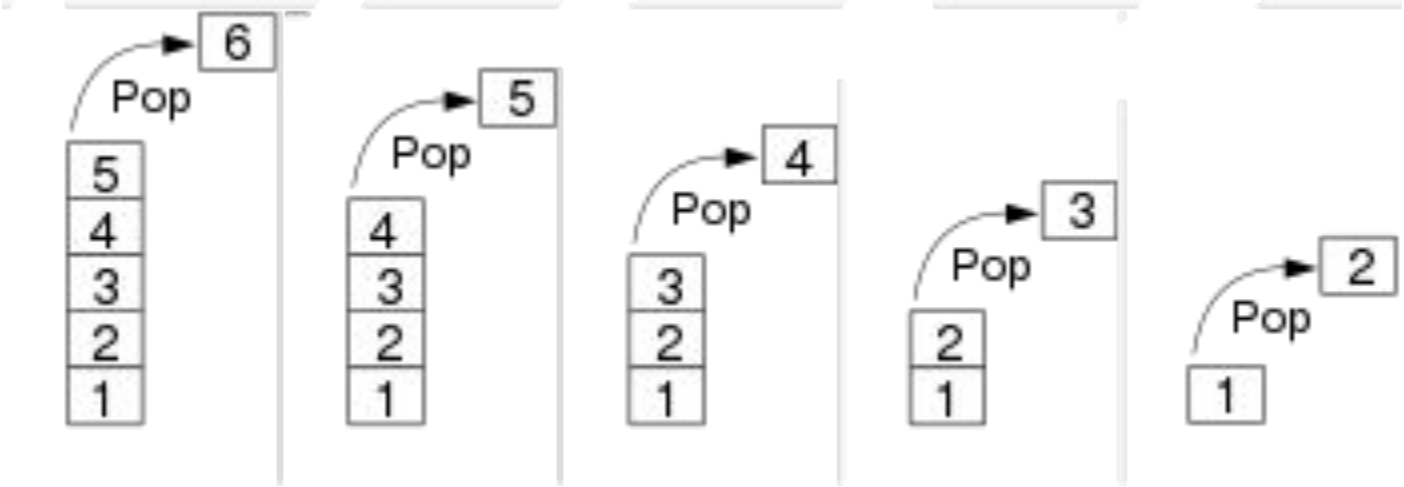
Let’s understand the difference between them two:-



Remove Items: pop

Newton School of Technology

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

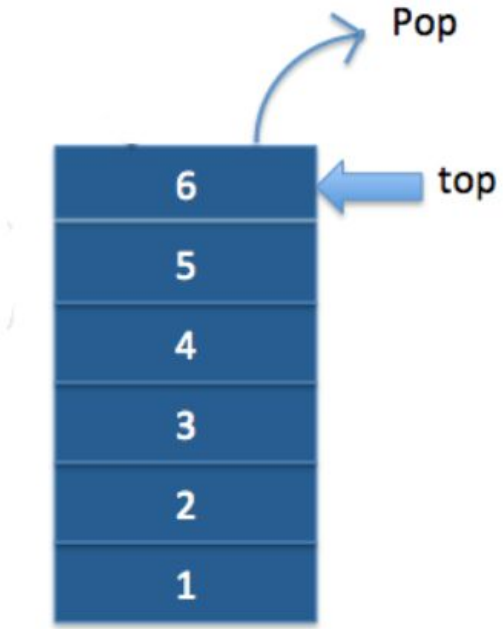


Remove Items: pop

Newton School of Technology

Let’s write javascript code for it:-

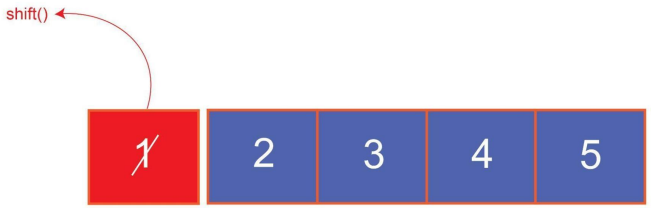
```
// Initial array with 5 items
let numbers = [1, 2, 3, 4, 5];

// Using pop to remove all items
numbers.pop();  // [1, 2, 3, 4]
numbers.pop();  // [1, 2, 3]
numbers.pop();  // [1, 2]
numbers.pop();  // [1]
numbers.pop();  // []

console.log(numbers); // []
```



Remove Items: shift

Newton School of Technology

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

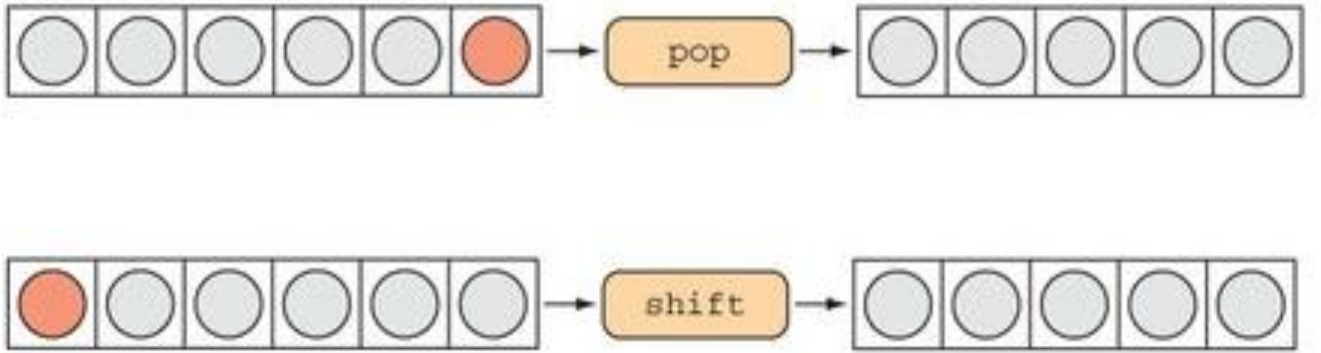



```
// Initial array with 5 items
let numbers = [1, 2, 3, 4, 5];

// Using shift to remove items
numbers.shift();  // [2, 3, 4, 5]
numbers.shift();  // [3, 4, 5]
numbers.shift();  // [4, 5]
numbers.shift();  // [5]
numbers.shift();  // []

console.log(numbers); // []
```

Difference: pop and shift

Newton School of Technology

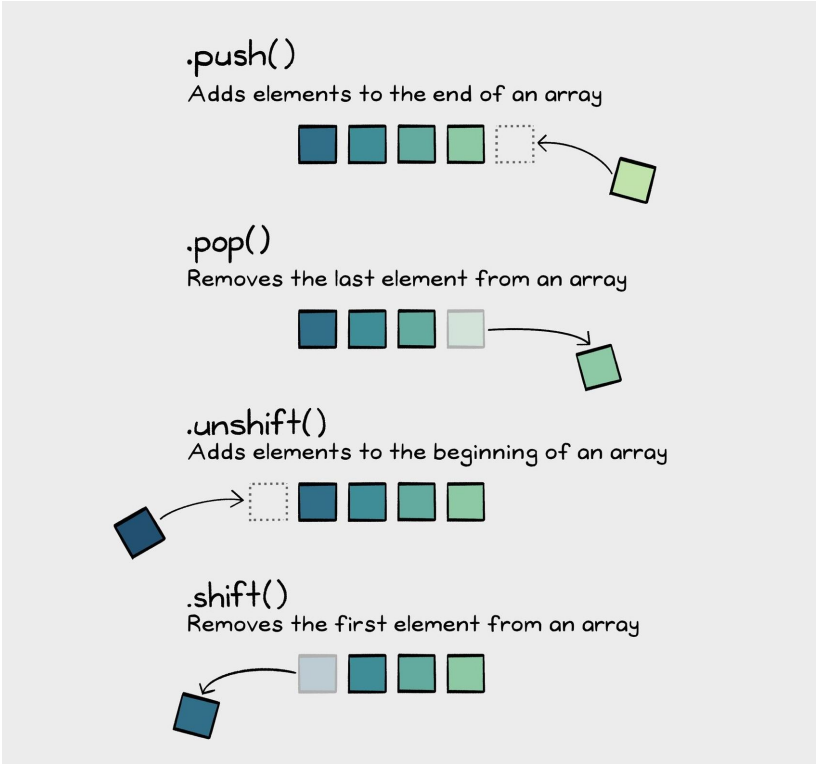
Let’s understand the difference between these two:-



Let’s quickly recap what we learned

Newton School of Technology

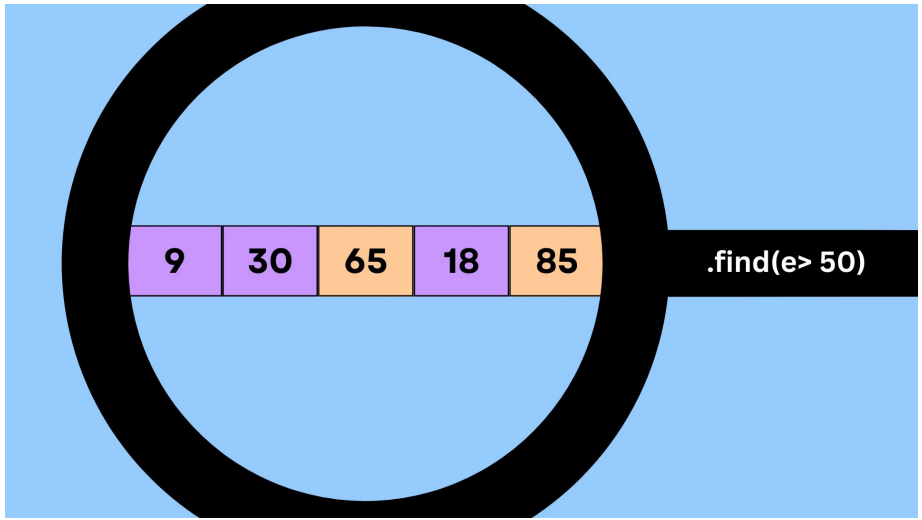
Here’s a quick recap of the array methods we've covered:



Search Methods: find()

Newton School of Technology

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.



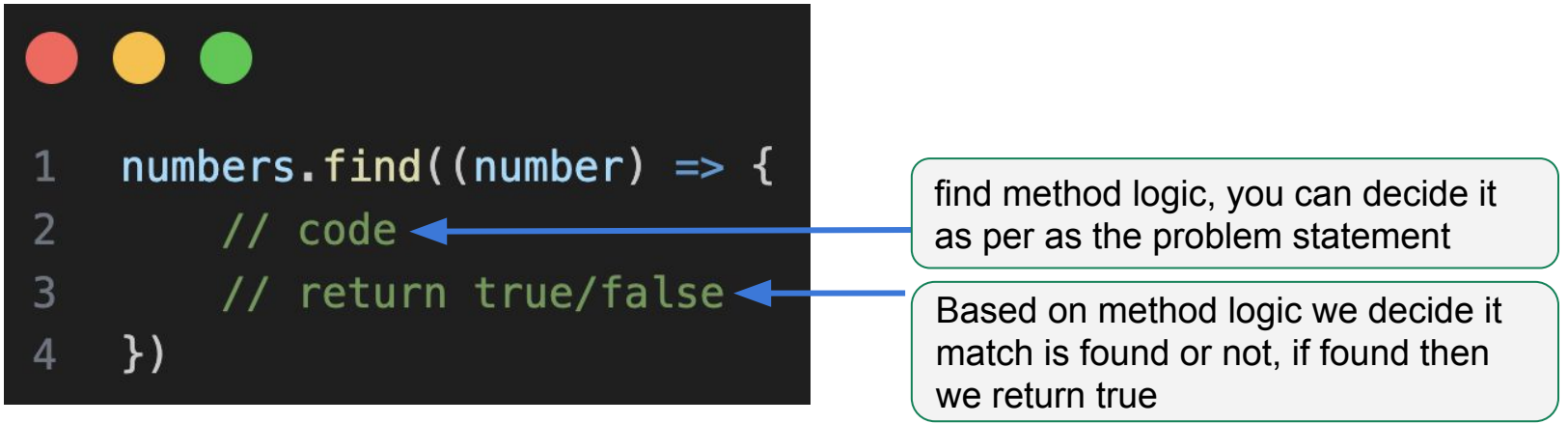
Since only two values i.e. 65 and 85 is greater than 50, but since 65 is the first occurrence, 65 gets returned.

find(): Syntax

Newton School of Technology

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

Syntax:



find() example 1

Newton School of Technology

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:-

```
// Define the array of numbers
const numbers = [5, 8, 12, 3, 7];

// `find` method to get the first number greater than 10
const greaterThanTen = numbers.find(num => {
  // Check if the current number is greater than 10
  const greater = num > 10;
  return greater;
});

// Log the result to the console
console.log(greaterThanTen); // Output: 12
```

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

find() example 2

Newton School of Technology

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:-

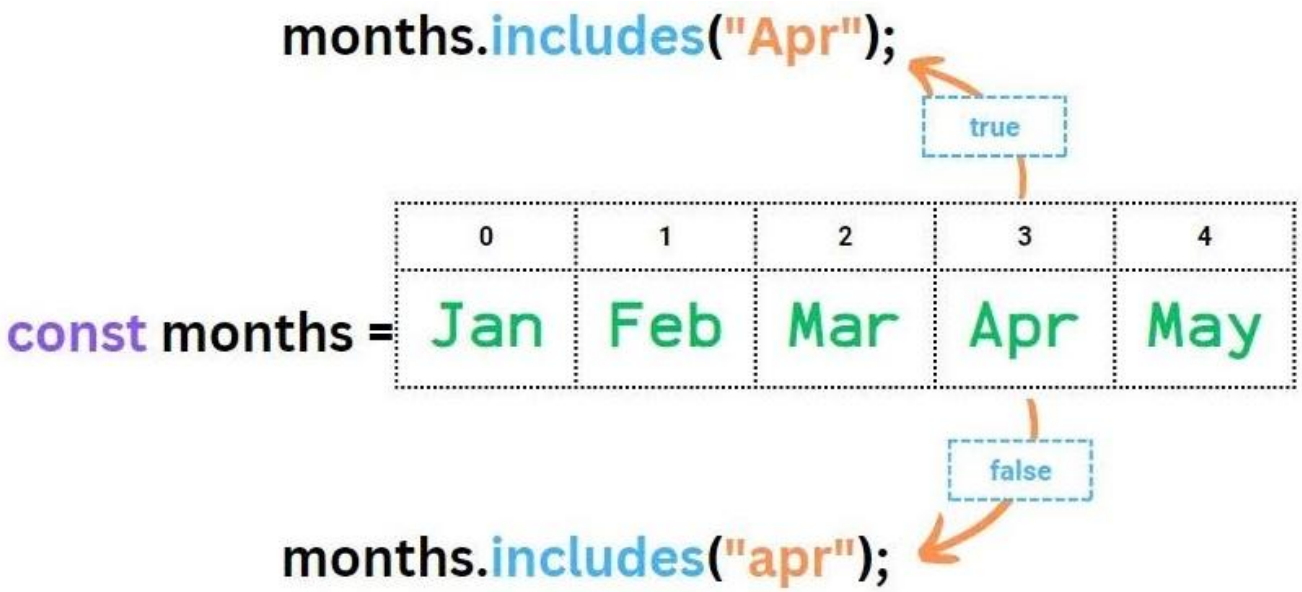
```
// Array of objects with name and age
let people = [
    { name: "John", age: 25 },
    { name: "Alice", age: 30 },
    { name: "Bob", age: 22 }
];

// find the first person who is 30 years old
let person = people.find(p => p.age === 30);

// Printing the result
console.log(person);
// Output: { name: "Alice", age: 30 }
```

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

Search Methods: includes()

Newton School of Technology

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.



includes() example

Newton School of Technology

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 the second instance.

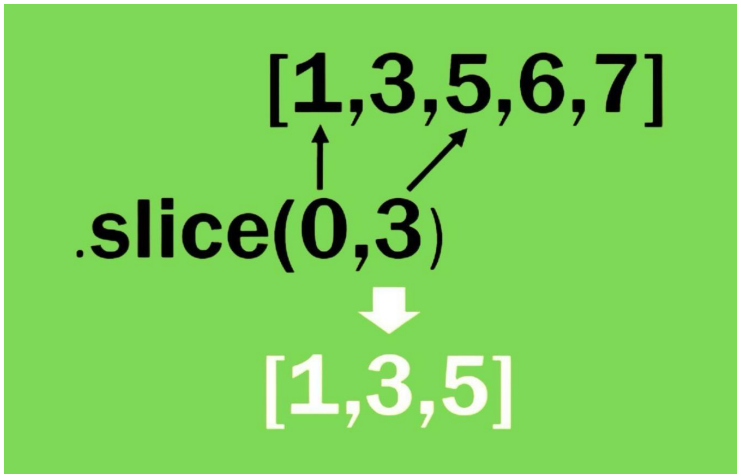
Newton School of Technology

Copying the array and modifying it directly

Copying the Array: slice() method

Newton School of Technology

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

Newton School of Technology

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

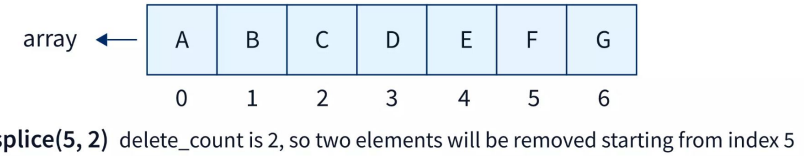
```
1  // Initial array
2  let numbers = [1, 3, 5, 6, 7];
3
4  // Using slice from index 0 to 3 (excluding index 3)
5  let slicedArray = numbers.slice(0, 3);
6
7  console.log(slicedArray); // [1, 3, 5]
8  console.log(numbers);
9  // [1, 3, 5, 6, 7] (Original array)
```

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

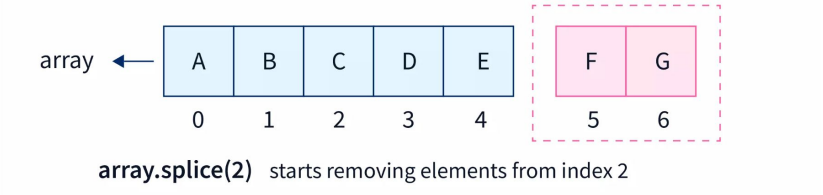
Modifying the Array: splice() method

Newton School of Technology

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

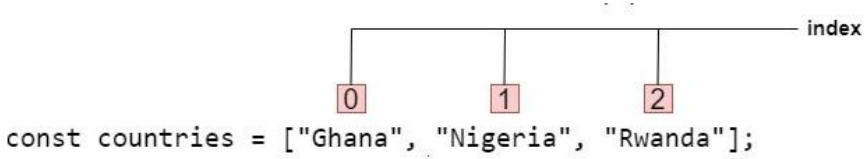


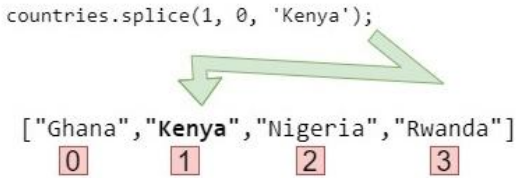
array.splice(5, 2) delete_count is 2, so two elements will be removed starting from index 5



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

Removing Items





Adding Items

splice(): Example

Newton School of Technology

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

```
// Initial array
let numbers = [1, 2, 3, 4, 5];

// Using splice to remove the first three items
numbers.splice(0, 3);

console.log(numbers); // [4, 5]
```

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

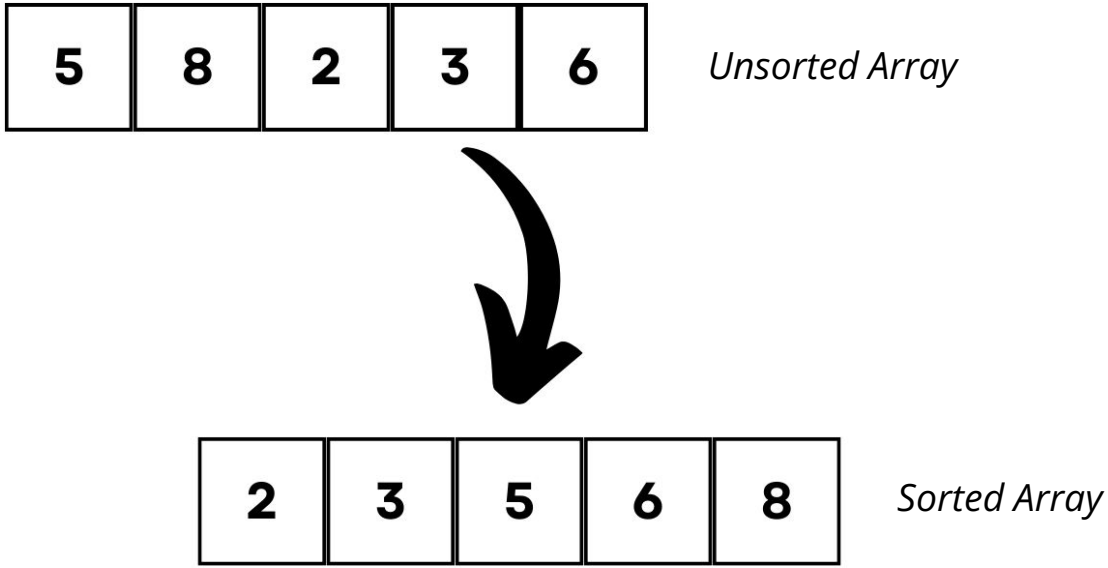
Newton School of Technology

Sort method

Organizing array: sort()

Newton School of Technology

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



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

sort(): How it works?

Newton School of Technology

By default sort() method sorts in alphabetical and 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 it's default behaviour.

sort(): How it works?

Newton School of Technology

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

```
// Array declaration and assignment
const array = [5, 4, 3, 2, 6];

// Sorting the array
array.sort(compareFunction);
```

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

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

sort(): Example Ascending

Newton School of Technology

Let’s sort the array in ascending order:-

```
// Sorting the array
array.sort(compareFunction);

// compare function ascending
function compareFunction(a, b){
    return a - b;
}

console.log(array);
// 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 is a - b **is negative** (a < b)

sort(): Example Descending

Newton School of Technology

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):

Newton School of Technology

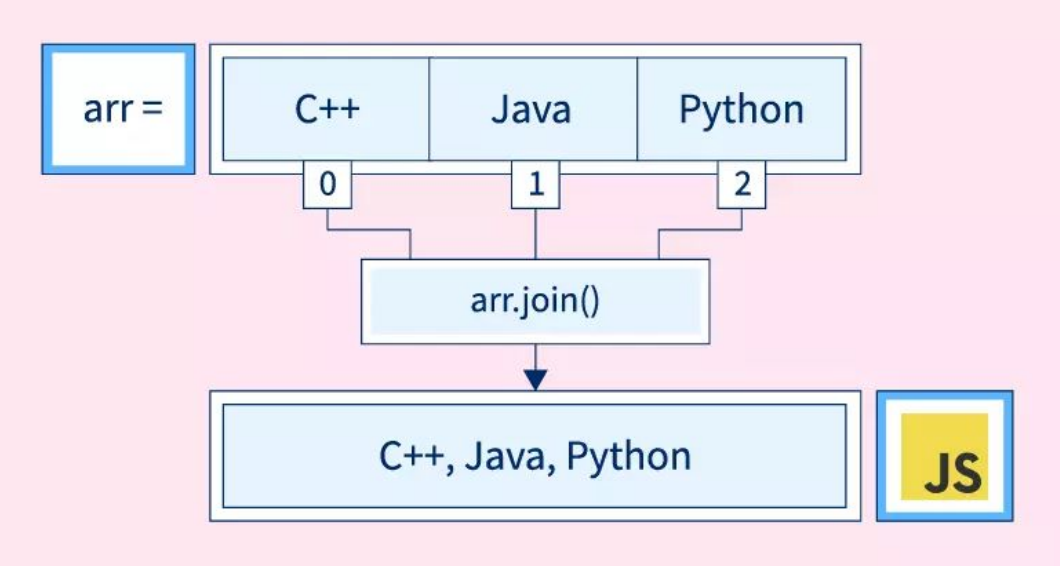
Bonus:

split and join methods

Array to String: join() method

Newton School of Technology

The 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

Newton School of Technology

Let’s understand join method with few examples:-

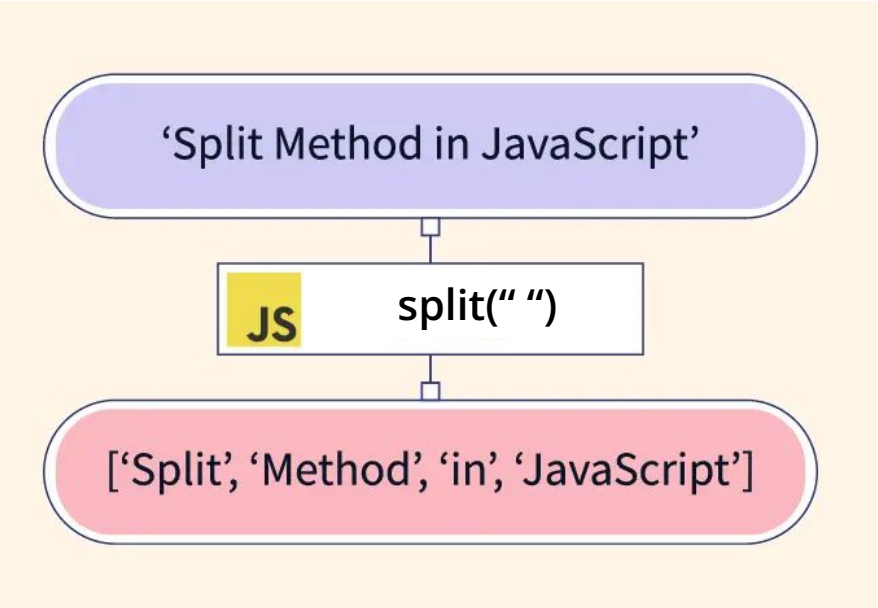
```
1  const languages = ['C++', 'Java', 'Python'];
2
3  // Using the default separator (comma)
4  console.log(languages.join());
5  // Output: "C++,Java,Python"
6
7  // Using a custom separator
8  console.log(languages.join(' | '));
9  // Output: "C++ | Java | Python"
10
11 // Joining without any separator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

Newton School of Technology

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

Newton School of Technology

Let’s understand split method with few examples:

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

```
const text = "C++,Java,Python";
const result = text.split();
// No separator provided

console.log(result);
// Output: ["C++,Java,Python"]
```

Here we splitted the array using whitespace a delimiter

```
const sentence = "Split method in javascript";
const wordsArray = sentence.split(" ");
// Splits the string at each space

console.log(wordsArray);
// Output: ["Split", "method", "in", "javascript"]
```

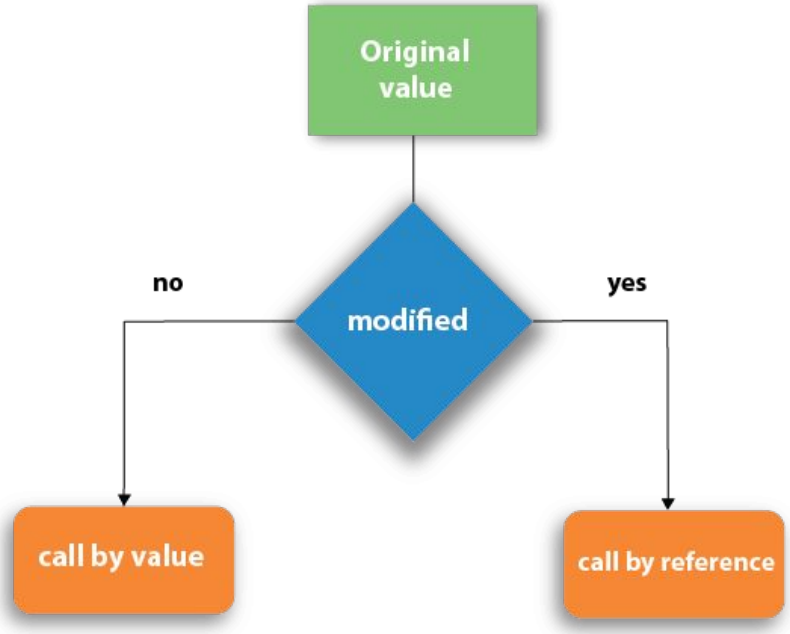
Newton School of Technology

Call by value vs call by reference

Call by Value vs Call by Reference

Newton School of Technology

Understanding how JavaScript handles primitives and reference types in functions

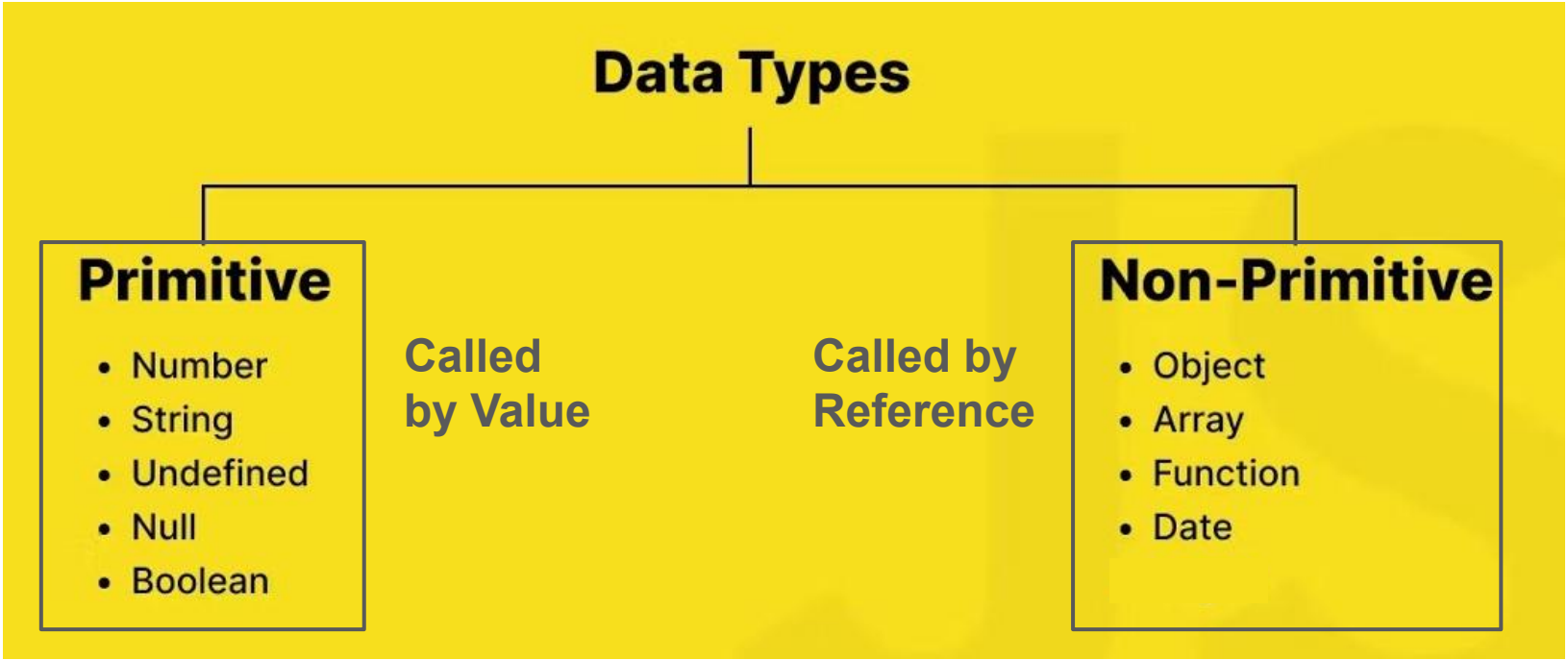


Understood?? Don’t worry, let’s jump into next slides and then come back, to understand.

Primitive vs Non-primitive

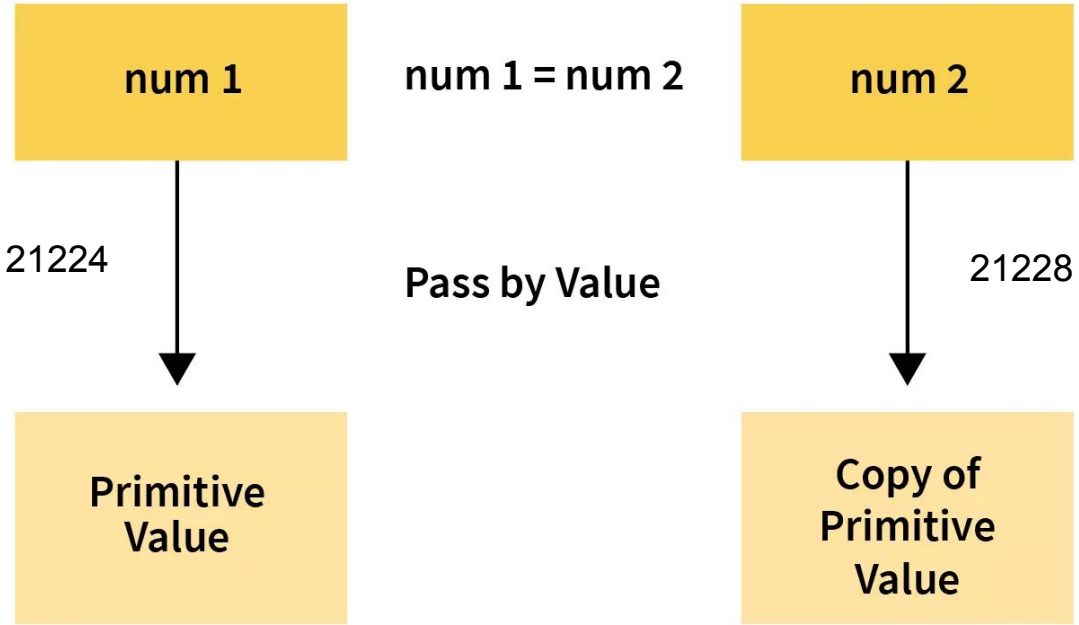
Newton School of Technology

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.

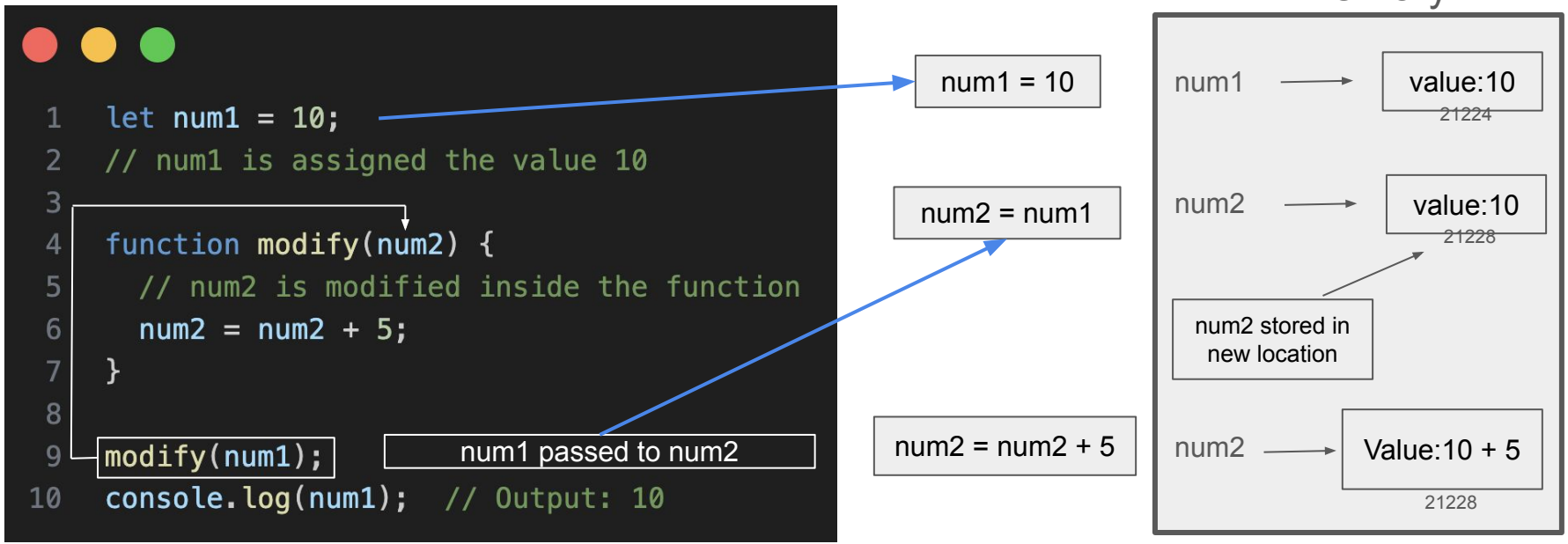


Here 21224 and 21228 are memory locations where num1 and num2 are stored respectively.

Call by Value: example

Newton School of Technology

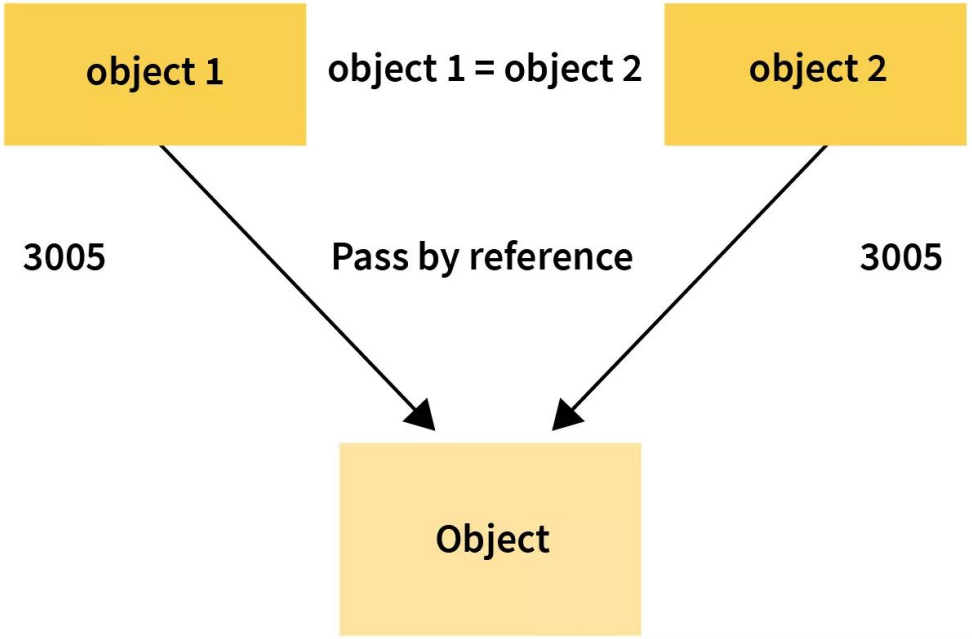
Let’s understand how it happens with an example.



Call by Reference

Newton School of Technology

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



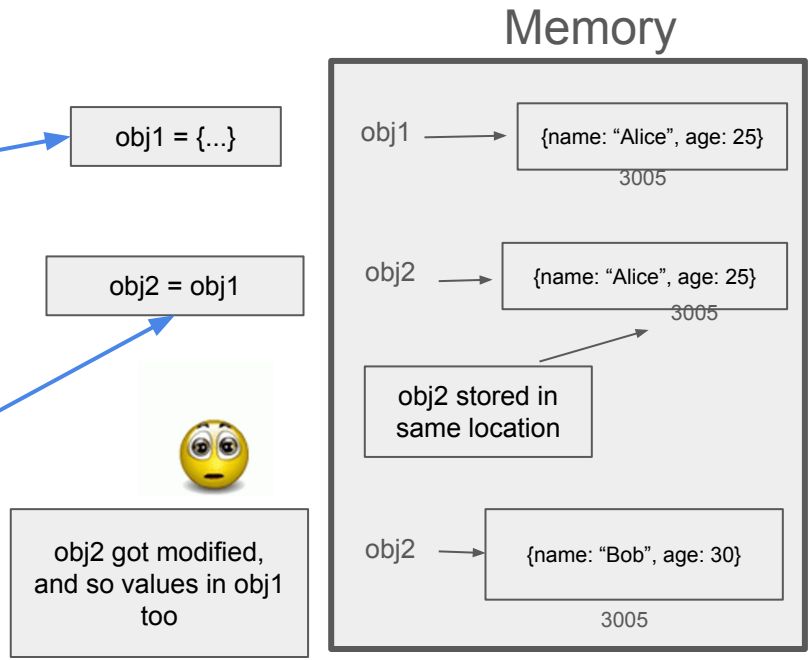
3005 is their common memory location.

Call by Reference: example

Newton School of Technology

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

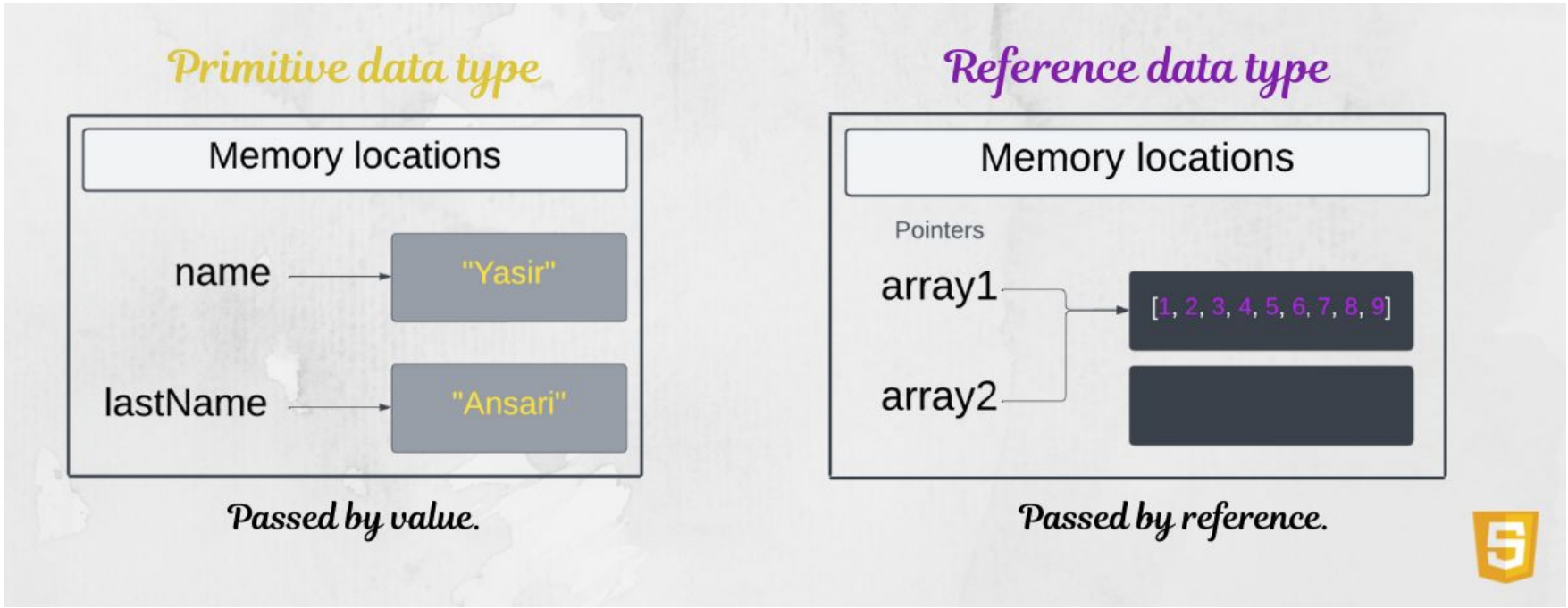
```
1  // Initial object
2  let object1 = { name: "Alice", age: 25 };
3
4  function modifyObject(object2) {
5    object2.age = 30;
6    // Modifies the original object via object2
7    object2.name = "Bob";
8    // Modifies the name property as well
9  }
10
11 // Passing object1 to the function
12 modifyObject(object1);
```



Quick Recap

Newton School of Technology

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



In Class Questions

Newton School of Technology

Thanks for watching!