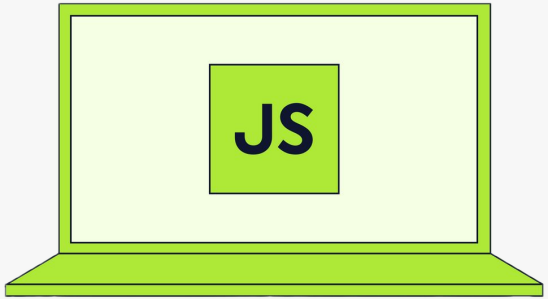
Newton School of Technology

# 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

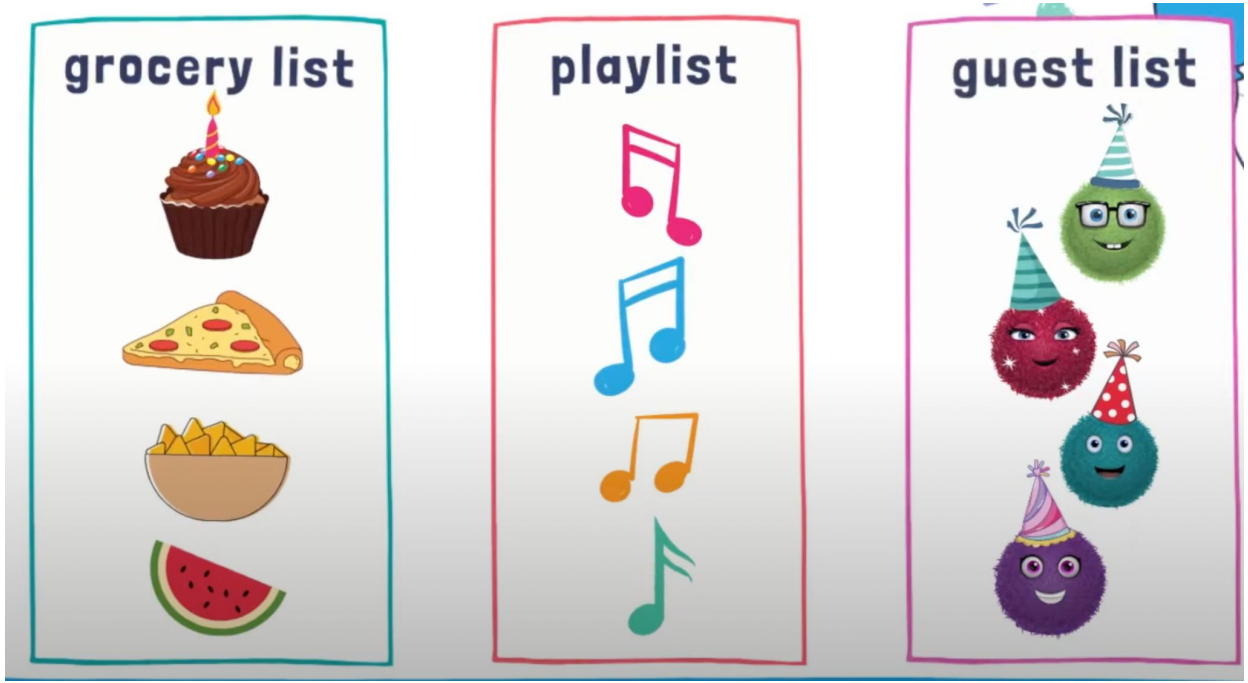
Imagine you are organizing a party, for that you need to purchase lot of items.


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:-

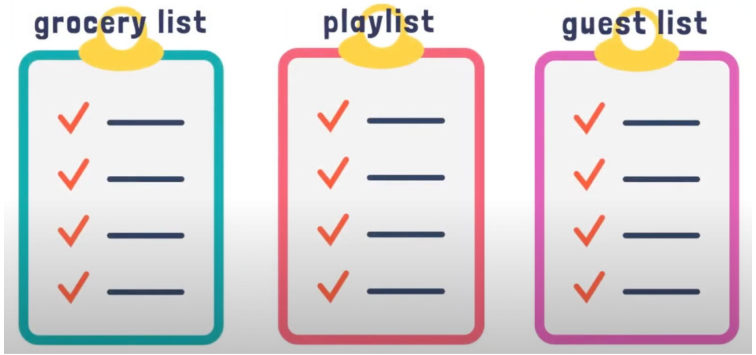


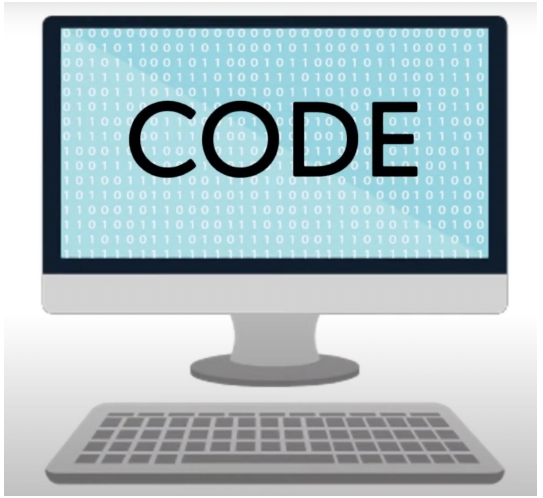

Lists help us to keep our tasks organized!!!

# 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

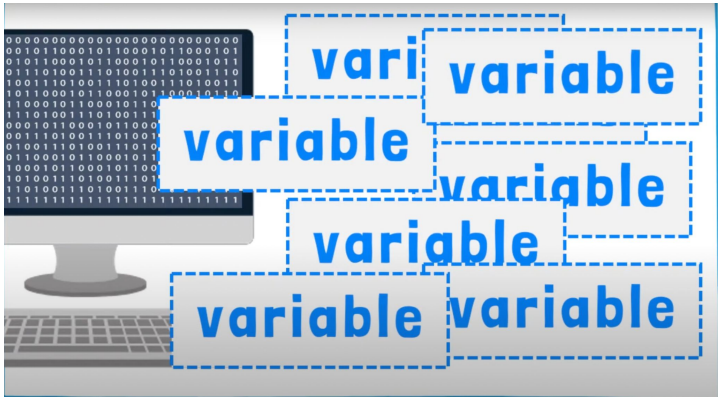
Lists in javascript are called “Array”.

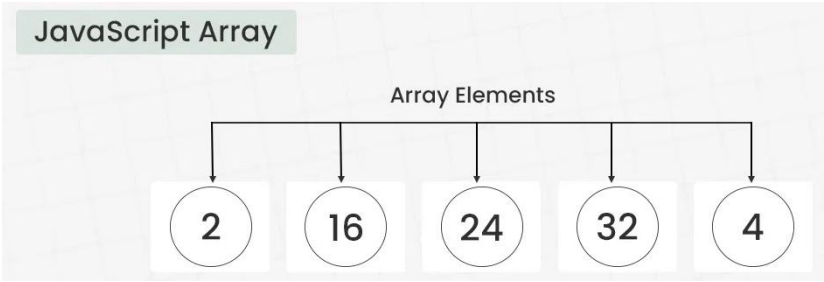
**Definition**: An array in JavaScript is a special object used to store multiple values in a single variable.

# 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.

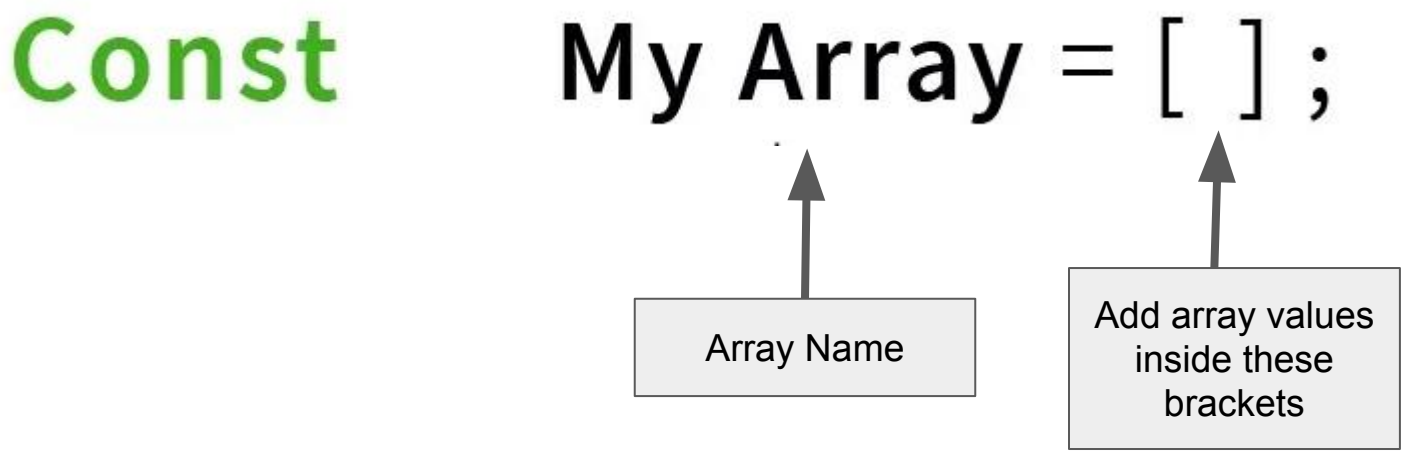



# Array: Definition and Syntax

Newton School of Technology

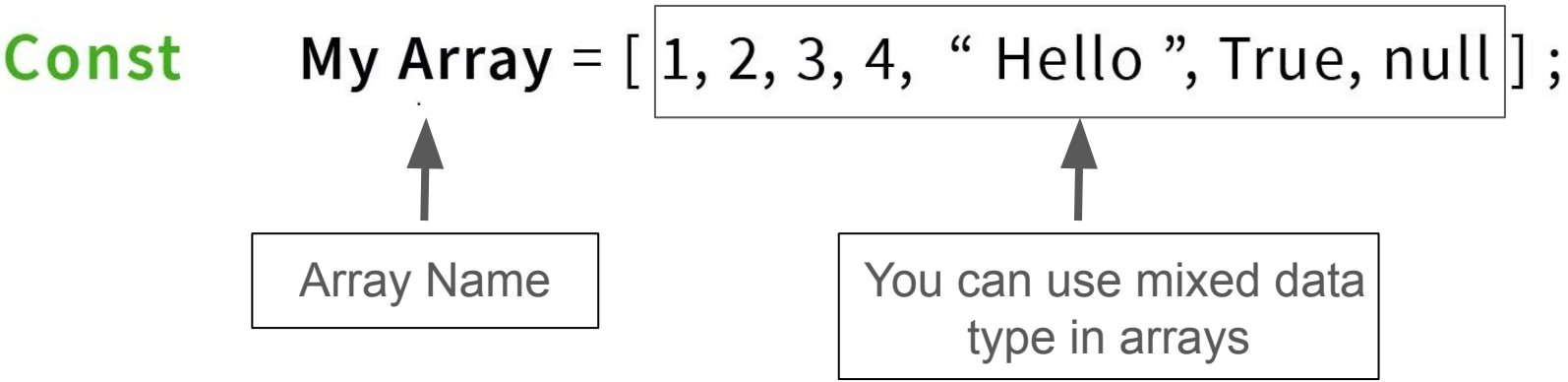
An array in JavaScript is a special object used to store multiple values in a single variable.



# Array: Storing values in arrays

Newton School of Technology

You can create array using either let or const keyword. We don't use var anymore:-



# 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

# Javascript arrays vs Python Lists

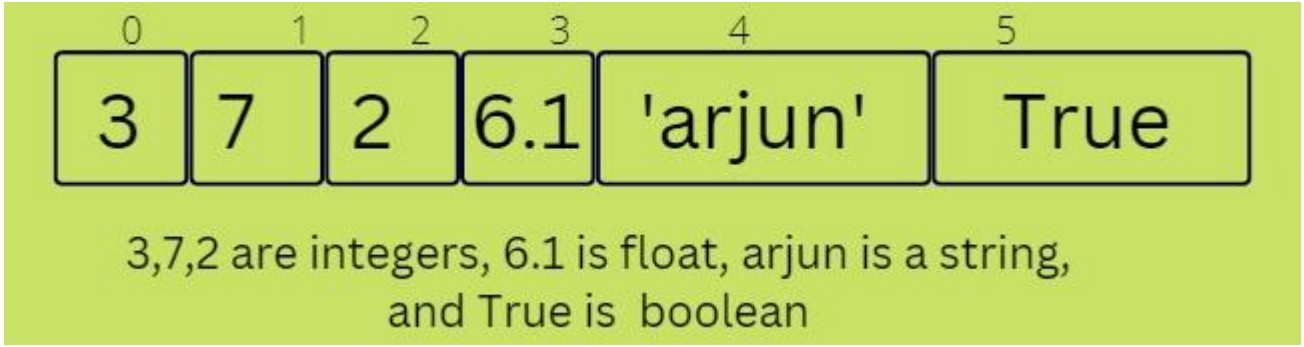
Newton School of Technology

Understanding their similarities and key differences for efficient coding.

# Similarities between two

Newton School of Technology

Both are **dynamic** and can store **mixed data types**.




Both can store these values.

# Javascript arrays as object

Newton School of Technology

In JavaScript, arrays are just specialized objects. Both arrays and regular objects are derived from the Object constructor.

```
let arr = new Array();
console.log(arr); // Output: []
```

Declaring array using Array Constructor

```
let arr = [1, 2, 3];

// Using built-in methods
arr.push(4); // Adds an element at the end
arr.pop(); // Removes the last element

// Adding a custom method
arr.sum = function() {
    return this.reduce((a, b) => a + b, 0);
};
console.log(arr.sum()); // Output: 6
```

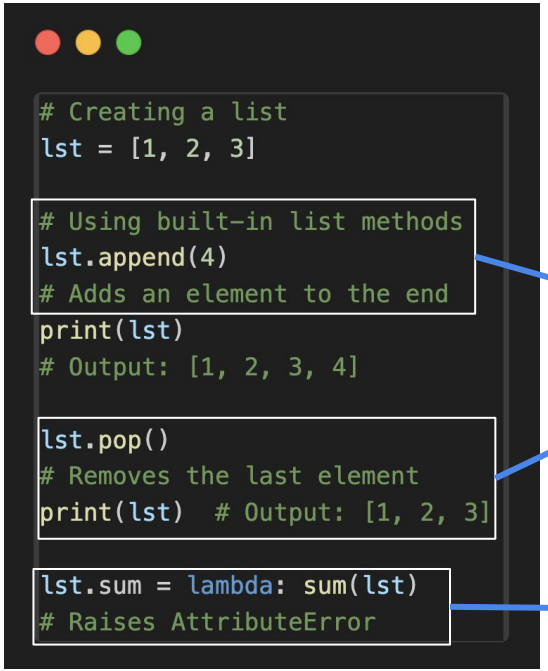
We can add functions just like we do in objects

In-built and user defined functions

# Python lists are not objects

Newton School of Technology

Python lists are not objects in the same sense as JavaScript arrays. They do not support adding custom attributes or methods directly.



```
# Creating a list
lst = [1, 2, 3]

# Using built-in list methods
lst.append(4)
# Adds an element to the end
print(lst)
# Output: [1, 2, 3, 4]

lst.pop()
# Removes the last element
print(lst)  # Output: [1, 2, 3]

lst.sum = lambda: sum(lst)
# Raises AttributeError
```

Lists are in-built data type i.e. we don't a constructor for it.

Lists too have in-built methods

But we can't add custom functions

Newton School of Technology

# Array Manipulation

# Array manipulation

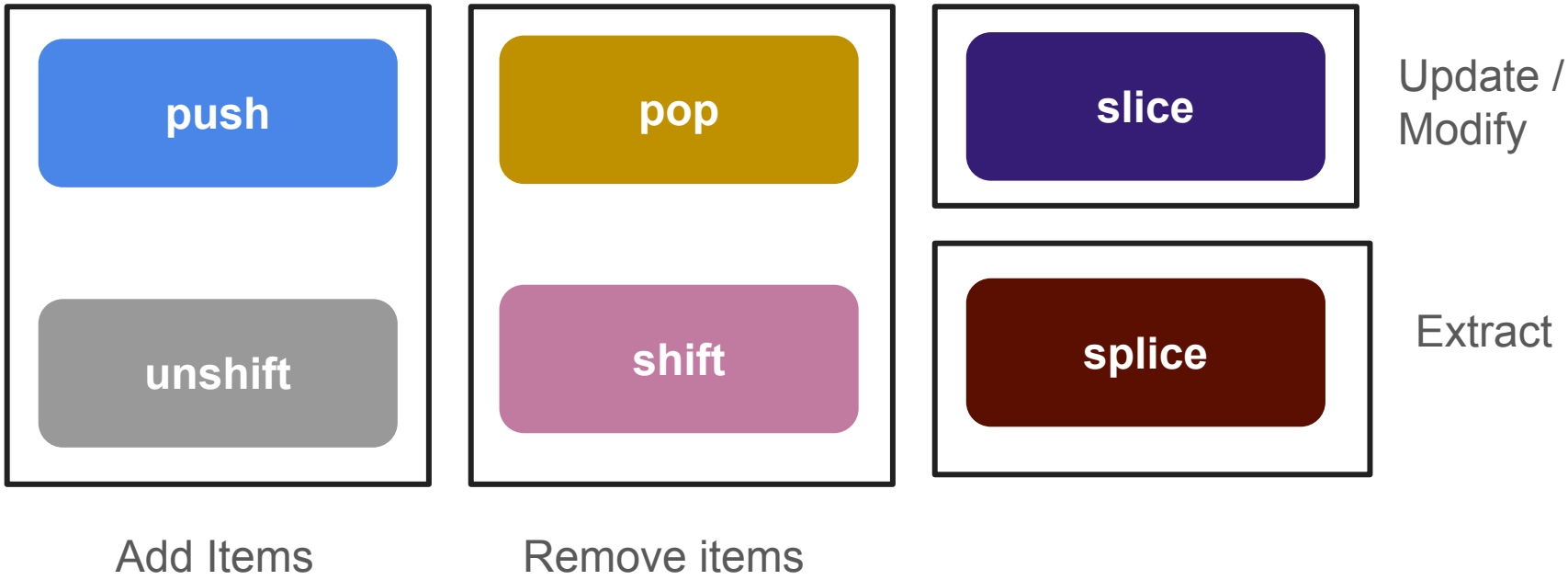
Newton School of Technology

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:

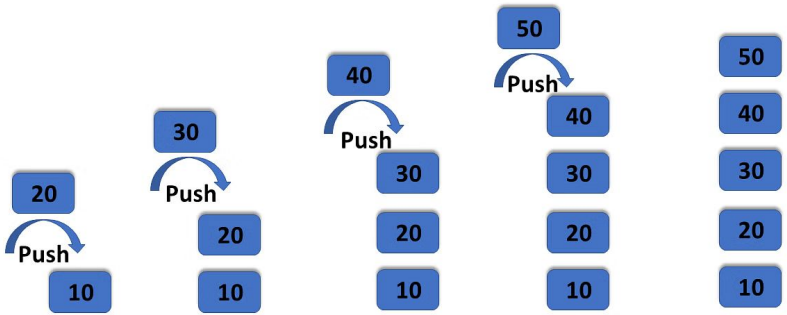


# Add Items: push

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

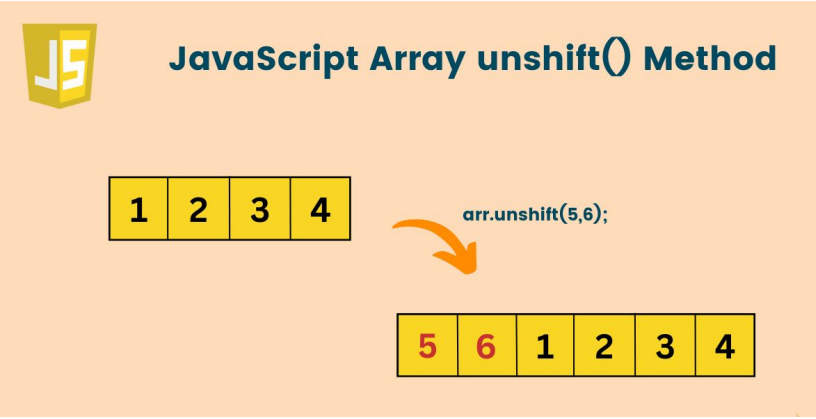
```
// Declaring an empty array
let numbers = [10];

// 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.

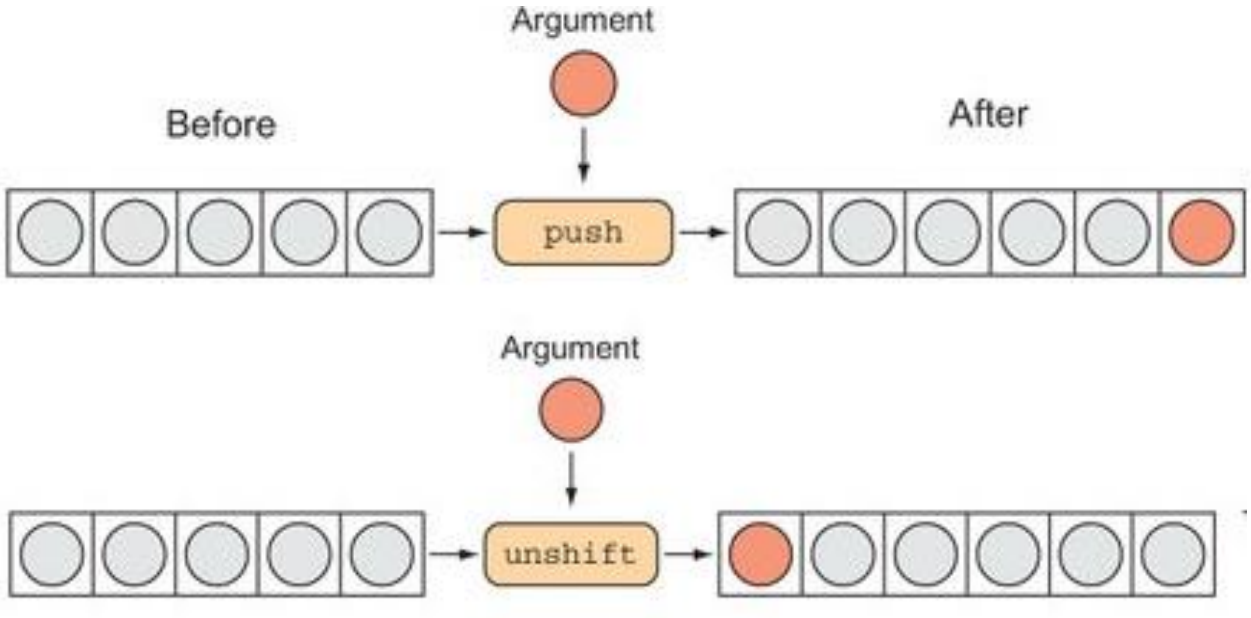



```
// Declaring an empty array
let numbers = [];

// 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

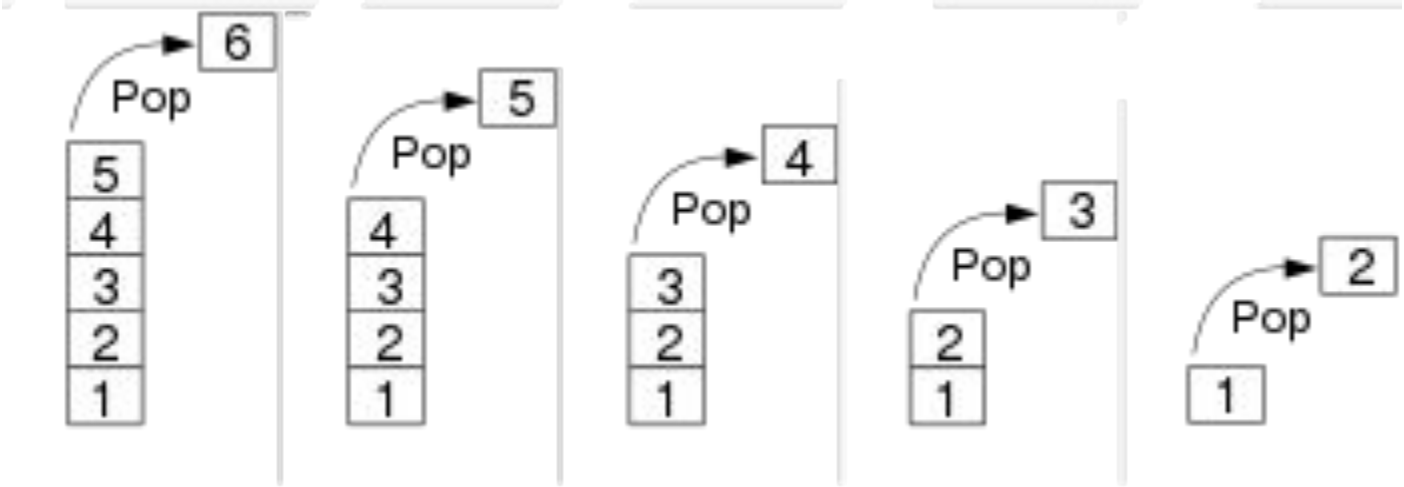
Let’s understand the difference between them two:-



# Remove Items: pop

Newton School of Technology

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

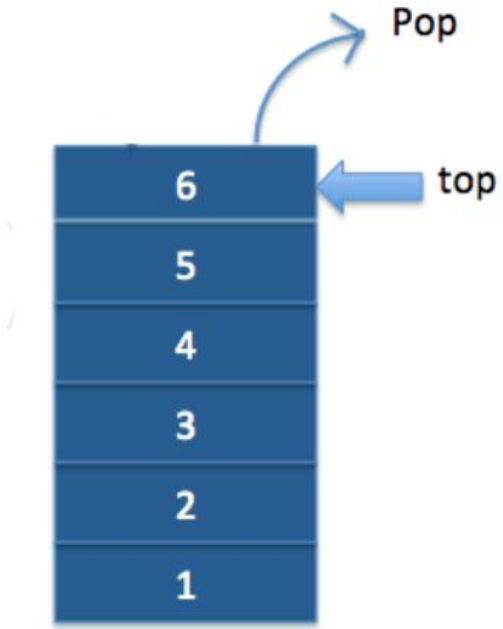


# Remove Items: pop

Newton School of Technology

Let’s write javascript code for it:-

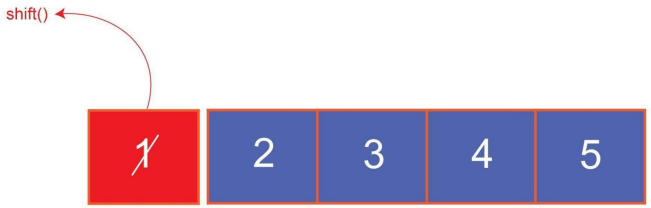
```
// Initial array with 5 items
let numbers = [1, 2, 3, 4, 5];

// 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.

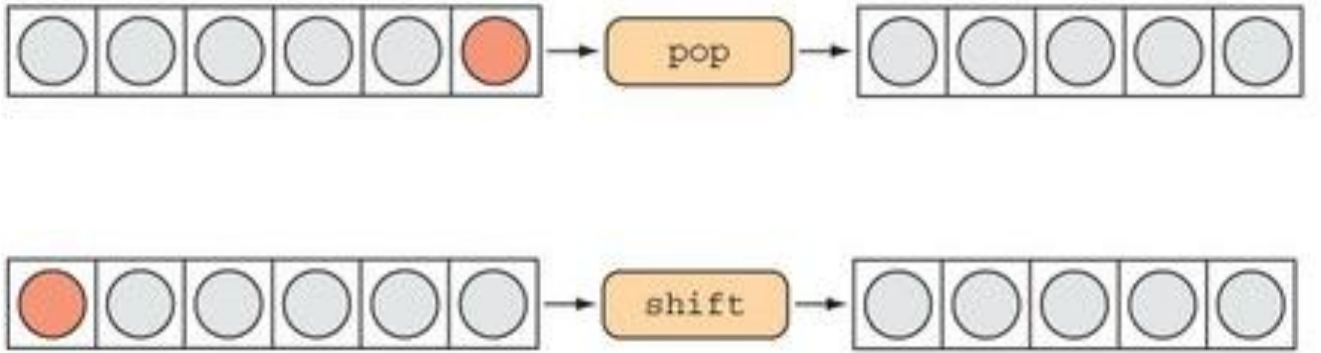



```
// Initial array with 5 items
let numbers = [1, 2, 3, 4, 5];

// 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

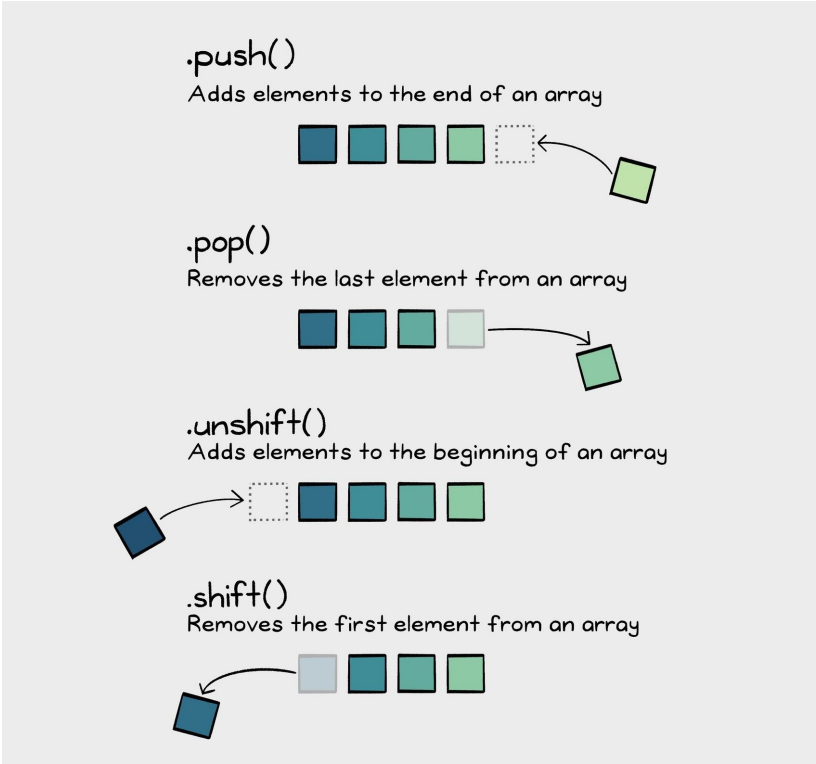
Let’s understand the difference between these two:-



# Let’s quickly recap what we learned

Newton School of Technology

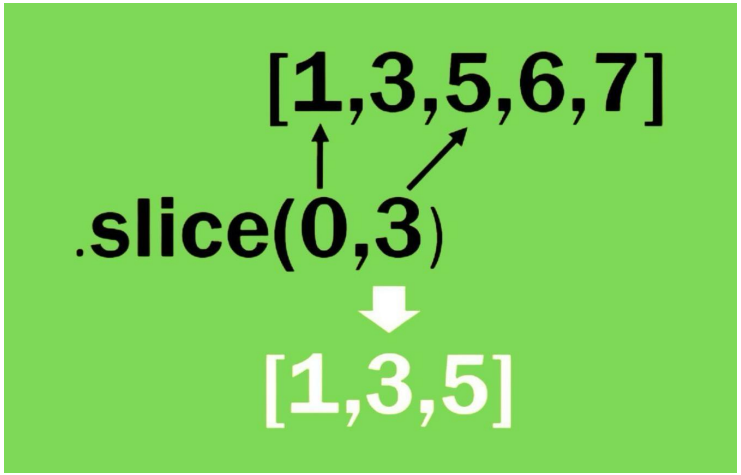
Here’s a quick recap of the array methods we've covered:



# Update/Modify: 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.

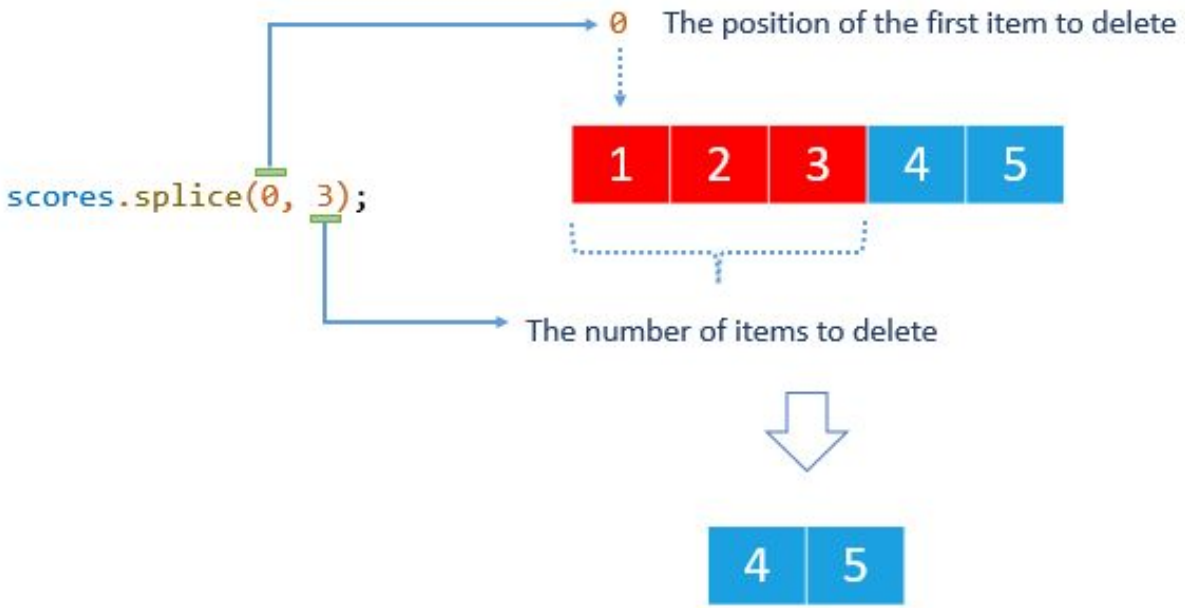
```
1  // Initial array
2  let numbers = [1, 3, 5, 6, 7];
3
4  // Using slice from index 0 to 3 (excluding index 3)
5  let slicedArray = numbers.slice(0, 3);
6
7  console.log(slicedArray); // [1, 3, 5]
8  console.log(numbers);
9  // [1, 3, 5, 6, 7] (Original array)
```

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

# Extract: 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.



# splice(): Example

Newton School of Technology

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

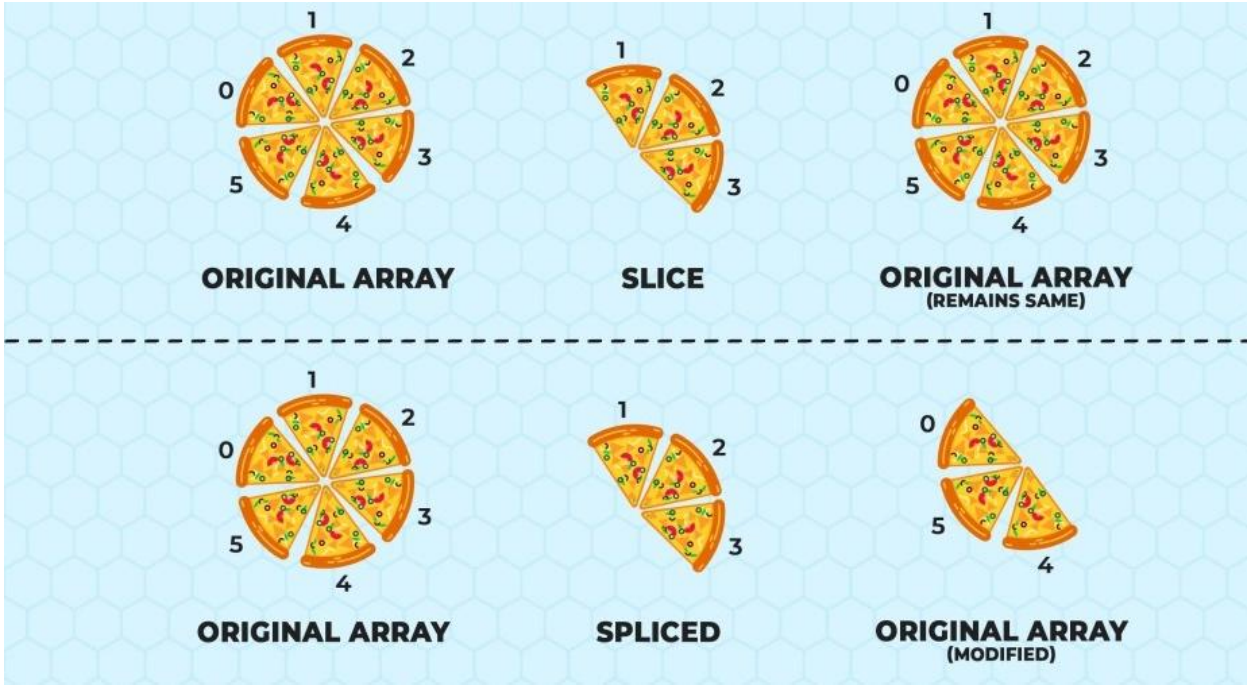
```
// Initial array
let numbers = [1, 2, 3, 4, 5];

// 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.

# Difference: slice() and splice() method

Newton School of Technology

The only difference is that slice method doesn't modifies the original array while splice method does modifies the original array.




Newton School of Technology

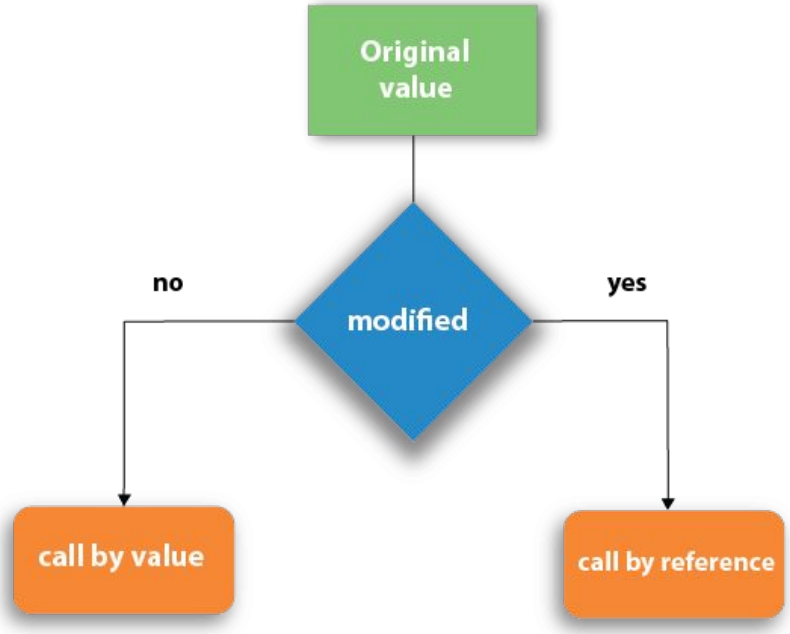
# Arrays:

# 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

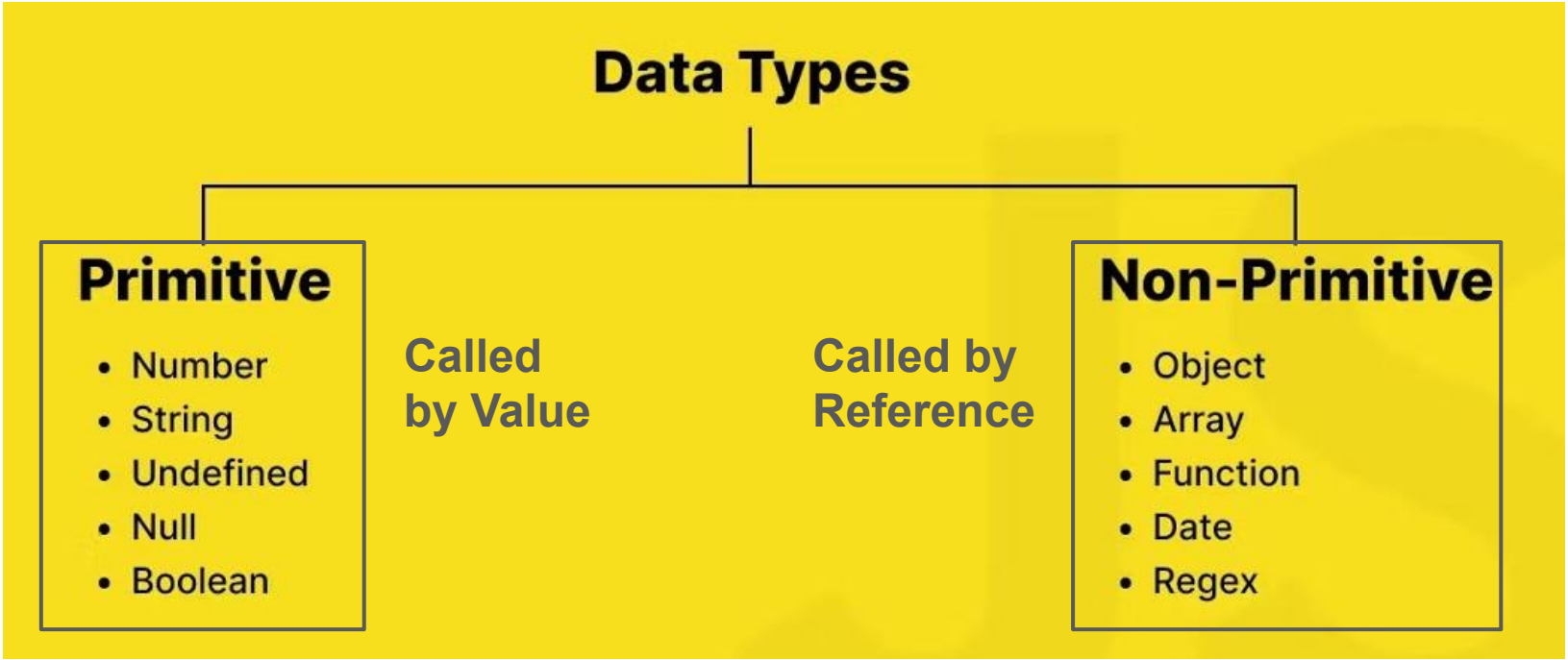



Understood?? Don’t worry, let’s jump into next slides and then come back, to understand.

# Primitive vs Non-primitive

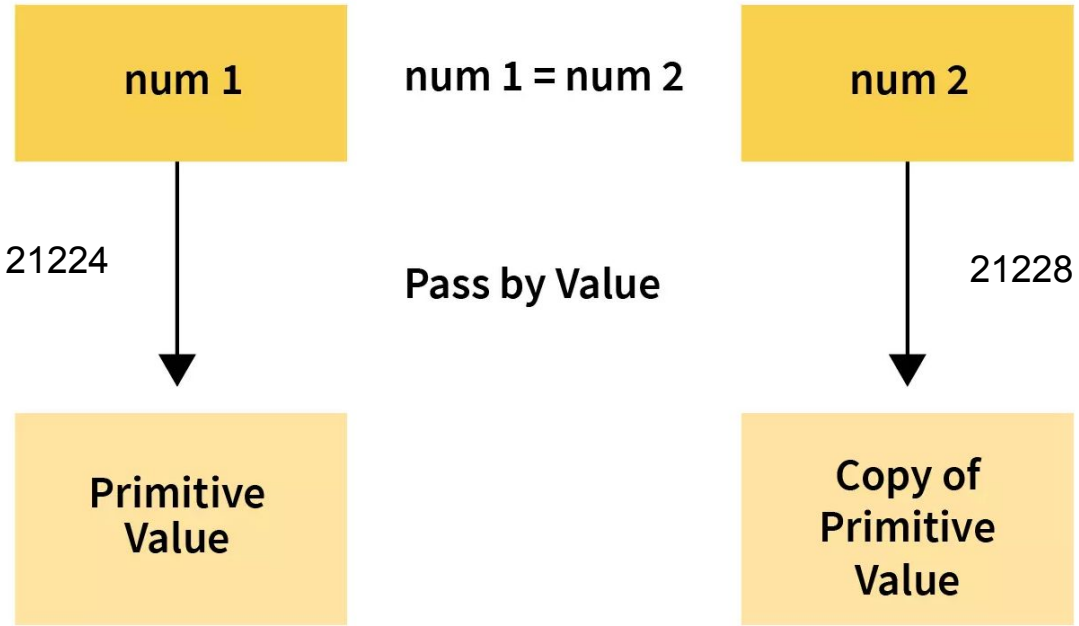
Newton School of Technology

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

Let’s understand how it happens with an example.

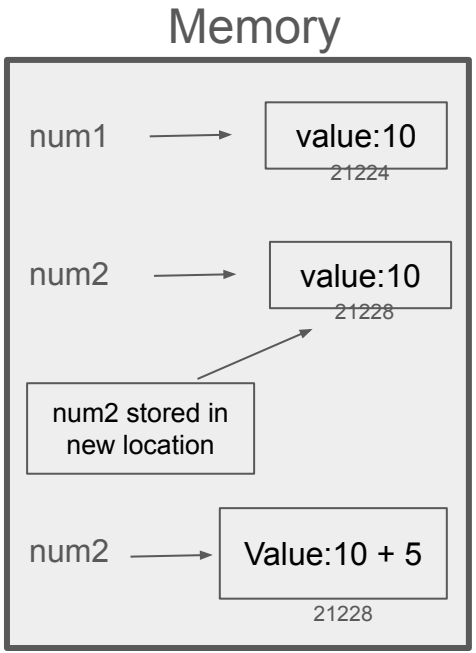
```
1   let num1 = 10;
2   // num1 is assigned the value 10
3
4   function modify(num2) {
5     // num2 is modified inside the function
6     num2 = num2 + 5;
7   }
8
9   modify(num1);
10  console.log(num1);  // Output: 10
```

num1 passed to num2

num1 = 10

num2 = num1

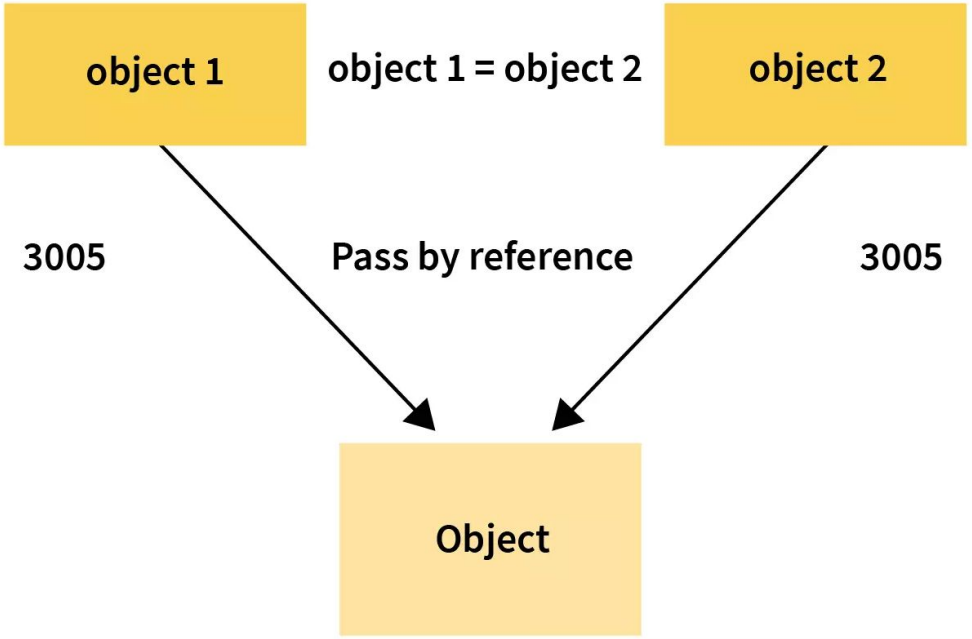
num2 = num2 + 5



# 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**.



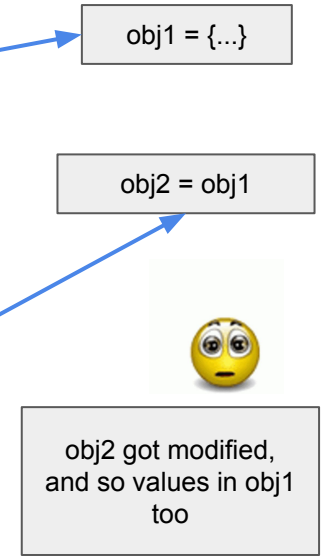

3005 is their common memory locations.

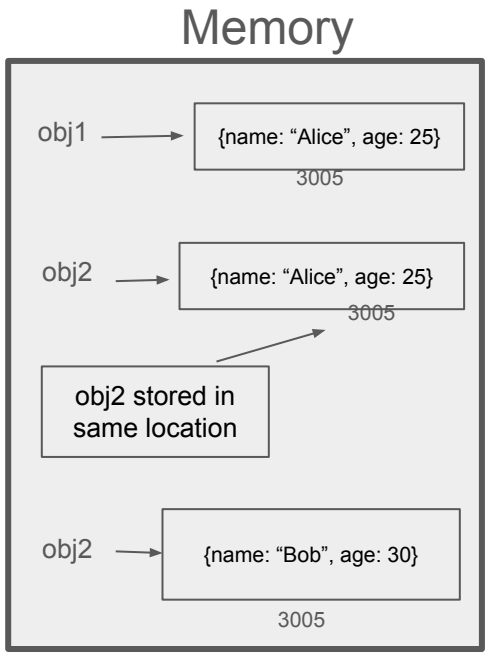
# Call by Reference: example

Newton School of Technology

Let’s understand how it happens with an example.

```
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);
```

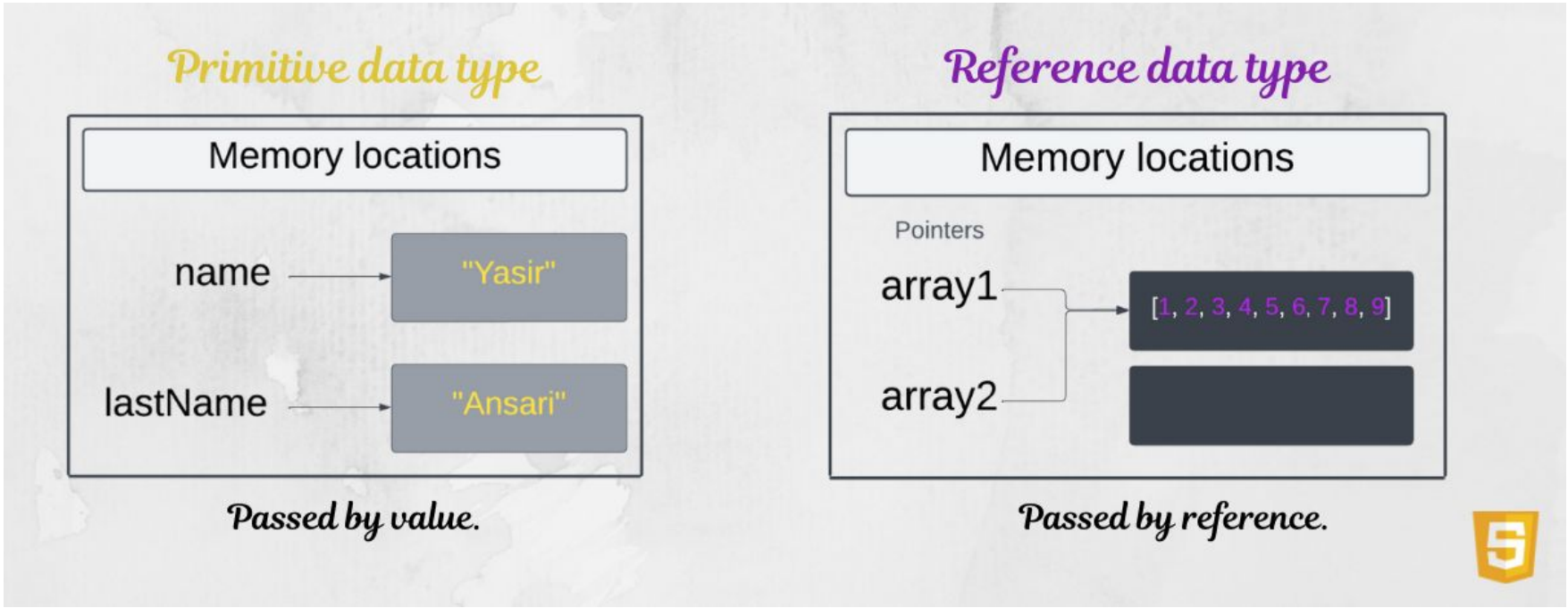


# Quick Recap

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!