[1] ARRAY AND METHODS

```
// Create some arrays
const numbers = [43,56,33,23,44,36,5];
const numbers2 = new Array(22,45,33,76,54);
const fruit = ['Apple', 'Banana', 'Orange', 'Pear'];
const mixed = [22, 'Hello', true, undefined, null, {a:1, b:1}, new Date()];

let val;

// Get array length
val = numbers.length;
// Check if is array
val = Array.isArray(numbers);
// Get single value
val = numbers[3];
val = numbers[0];
// Insert into array
numbers[2] = 100;
// Find index of value
val = numbers.indexOf(36);

console.log(numbers);
console.log(val);
```

```
// MUTATING ARRAYS
// Add on to end
numbers.push(250);
// Add on to front
numbers.unshift(120);
// Take off from end
numbers.pop();
// Take off from front
numbers.shift();
// Splice values
numbers.splice(1,3);
// Reverse
numbers.reverse();

// Concatenate array
val = numbers.concat(numbers2);
```

Creating arrays

```
--Easy way
const numbers = [1.2.3] <-- Forma tradicional de crearlos.

--Instance
const moreNumbers = new Array();
const moreNumbers = new Array(5); <-- (Empty array. Length 5)
const moreNumbers = new Array('New', 'Item'))

--Similar to Instance
const otherNumbers = Array()
Array.of (1)

--Creating an array from an iterable or array-object
const moreNumbers = Array.from()
const moreNumbers = Array.from('Hi!')
```

Spread Syntax

Spread syntax (...) allows an iterable, such as an array or string, to be expanded in places where zero or more arguments (for function calls) or elements (for array literals) are expected. In an object literal, the spread syntax enumerates the properties of an object and adds the key-value pairs to the object being created. Spread syntax can be used when all elements from an object or array need to be included in a new array or object, or should be applied one-by-one in a function call's arguments list. There are three distinct places that accept the spread syntax:

- Function arguments list (myFunction(a, ...iterableObj, b))
- Array literals ([1, ...iterableObj, '4', 'five', 6])
- Object literals ({ ...obj, key: 'value' })

DM: [Spread syntax (...) - JavaScript | MDN (mozilla.org)]

```
---Syntax:
myFunction(a, ...iterableObj, b)
[1, ...iterableObj, '4', 'five', 6]
{ ...obj, key: 'value' }
```

Destructuring assignment syntax

The destructuring assignment syntax is a JavaScript expression that makes it possible to unpack values from arrays, or properties from objects, into distinct variables.

DM: [Destructuring assignment - JavaScript | MDN (mozilla.org)]

```
---Example[1]
let a, b, rest;
[a, b] = [10, 20];
console.log(a); //10
console.log(b); //20

---Example[2]
[a, b, ...rest] = [10, 20, 30, 40, 50];
console.log(rest); //[30,40,50]
```

Find() --- OBJECTS (aunque funciona con arrays también)

Returns the value of the first element that passes a test. Executes a function for each array element. Returns undefined if no elements are found. Does not execute the function for empty elements.

DM: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/find]

```
---Sintax:
array.find(function(currentValue, index, arr),thisValue)

---Arrow function
find((element) => { /* … */ } )
find((element, index) => { /* … */ } )
find((element, index, array) => { /* … */ } )

---Example[1]
const personData = [{name: 'max'}, {name: 'manuel'}]
const manuel = personData.find((person, idx, persons) => {
return person.name === 'manuel'
})

---Example[2]
const inventory = [
{ name: "apples", quantity: 2 },
{ name: "bananas", quantity: 0 },
{ name: "cherries", quantity: 5 },
];
function isCherries(fruit) {
return fruit.name === "cherries";
}
console.log(inventory.find(isCherries)); // { name: 'cherries', quantity: 5 }

const result = inventory.find(({ name }) => name === "cherries"); <-- same using an arrow
function.
```

```
console.log(result); // { name: 'cherries', quantity: 5 }
```

Includes

The includes() method determines whether an array includes a certain value among its entries, returning true or false as appropriate.
DM: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/includes

```
---Syntax:
includes(searchElement)
includes(searchElement, fromIndex)

---Example [1]
const array1 = [1, 2, 3];
console.log(array1.includes(2)); // True
```

Slice

The slice() method returns a shallow copy of a portion of an array into a new array object selected from start to end (end not included) where start and end represent the index of items in that array. The original array will not be modified. slice method can also be called to convert Array-like objects/collections to a new Array.
DM: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/slice

```
---Syntax:
slice()
slice(start)
slice(start, end)

---Example[1]
const fruits = ['Banana', 'Orange', 'Lemon', 'Apple', 'Mango'];
const citrus = fruits.slice(1, 3); // ['Orange','Lemon']
```

Concat

The `concat()` method is used to merge two or more arrays. This method does not change the existing arrays, but instead returns a new array. The concat() method is a copying method. It does not alter this or any of the arrays provided as arguments but instead returns a shallow copy that contains the same elements as the ones from the

original arrays. Concat does not treat all array-like objects as arrays by default — only if Symbol.isConcatSpreadable is set to a truthy value (e.g. true).

```
---Syntax:
concat()
concat(value0)
concat(value0, value1)
concat(value0, value1, /* … ,*/ valueN)

---Example [1]
const array1 = ['a', 'b', 'c'];
const array2 = ['d', 'e', 'f'];
const array3 = array1.concat(array2); // ["a", "b", "c", "d", "e", "f"]
```

Foreach()

The `forEach()` method executes a provided function once for each array element. Alternative for loops.

DM: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/forEach

```
---Syntax:
// Arrow function
forEach((element) => { /* … */ })
forEach((element, index) => { /* … */ })
forEach((element, index, array) => { /* … */ })
// Callback function
forEach(callbackFn)
forEach(callbackFn, thisArg)
// Inline callback function
forEach(function(element) { /* … */ })
forEach(function(element, index) { /* … */ })
forEach(function(element, index, array){ /* … */ })
forEach(function(element, index, array) { /* … */ }, thisArg)

---Example [1]
const items = ['item1', 'item2', 'item3'];
const copyItems = [];
// before
for (let i = 0; i < items.length; i++) {
copyItems.push(items[i]);
}
// after
items.forEach((item) => {
copyItems.push(item);
```

```
});
```

map()

map calls a provided callbackFn function once for each element in an array, in order, and constructs a new array from the results. The map() method is a copying method. It does not alter this.

DM: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map

```
---Syntax:
// Arrow function
map((element) => { /* … */ })
map((element, index) => { /* … */ })
map((element, index, array) => { /* … */ })
// Callback function
map(callbackFn)
map(callbackFn, thisArg)
// Inline callback function
map(function(element) { /* … */ })
map(function(element, index) { /* … */ })
map(function(element, index, array){ /* … */ })
map(function(element, index, array) { /* … */ }, thisArg)

---Example[1]
const numbers = [1, 4, 9];
const roots = numbers.map((num) => Math.sqrt(num)); // Roots is now [1,2,3]

---Example[2]
const kvArray = [
{ key: 1, value: 10 },
{ key: 2, value: 20 },
{ key: 3, value: 30 },
];
const reformattedArray = kvArray.map(({ key, value}) => ({ [key]: value })); //
reformattedArray is now [{1: 10}, {2: 20}, {3: 30}],
```

sort()

Sorts the elements of an array in place and returns the reference to the same array, now sorted. If `compareFn` is not supplied, all non-`undefined` array elements are sorted by converting them to strings and comparing strings in UTF-16 code units order.

DM: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/sort

```
---Syntax:
// Functionless
sort()
// Arrow function
sort((a, b) => { /* … */ } )
// Compare function
sort(compareFn)
// Inline compare function
sort(function compareFn(a, b) { /* … */ })

---Example[1]
const months = ['March', 'Jan', 'Feb', 'Dec'];
months.sort(); // ["Dec", "Feb", "Jan", "March"]
```

filter()

The filter() method creates a shallow copy of a portion of a given array (does not affected the original array) , filtered down to just the elements from the given array that pass the test implemented by the provided function.

DM: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter

```
---Syntax:
// Arrow function
filter((element) => { /* … */ } )
filter((element, index) => { /* … */ } )
filter((element, index, array) => { /* … */ } )
// Callback function
filter(callbackFn)
filter(callbackFn, thisArg)
// Inline callback function
filter(function(element) { /* … */ })
filter(function(element, index) { /* … */ })
filter(function(element, index, array){ /* … */ })
filter(function(element, index, array) { /* … */ }, thisArg)

---Example[1]
const words = ['spray', 'limit', 'elite', 'exuberant', 'destruction', 'present'];
const result = words.filter(word => word.length > 6); // ["exuberant", "destruction",
"present"]
```

reduce()

The reduce() method executes a user-supplied "reducer" callback function on each element of the array, in order, passing in the return value from the calculation on the preceding element. The final result of running the reducer across all elements of the array is a single value. The reduce() method itself does not mutate the array it is used on. However, it is possible for code inside the callback function to mutate the array (See dev notes).

DM: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/reduce

https://www.w3schools.com/jsref/jsref_reduce.asp

```
---Syntax:
array.reduce(function(total, currentValue, currentIndex, arr), initialValue)

--Example[1]
const array1 = [1, 2, 3, 4];
const initialValue = 0;
const sumWithInitial = array1.reduce(
(previousValue, currentValue) => previousValue + currentValue,
initialValue
); //Result is 10
```

join()

The `join()` method creates and returns a new string by concatenating all of the elements in an array (or an array-like object), separated by commas or a specified separator string. If the array has only one item, then that item will be returned without using the separator.

DM: Array.prototype.join() - JavaScript | MDN (mozilla.org)

```
--=Syntax:
join()
join(separator)

--Example[1]
const elements = ['Fire', 'Air', 'Water'];
console.log(elements.join()); //"Fire,Air,Water"
console.log(elements.join('')); //"FireAirWater"
```

```
console.log(elements.join('-')); //"Fire-Air-Water"
```