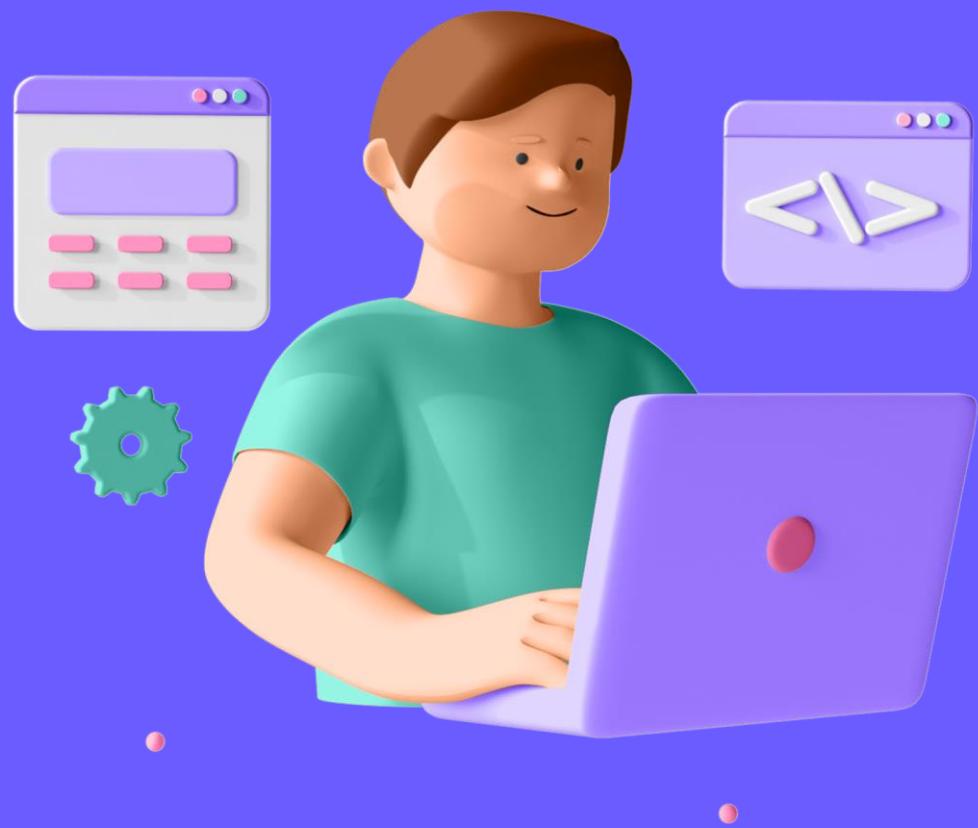




Somos un **ecosistema** de desarrolladores de software

Java Script



```
<!--  
n-->  
BEGIN NAVIGATION  
" >Home</a></li>  
.html">Home Events</a></li>  
nu.html">Multiple Column Men  
<a href="#" class="current":  
button-header.html">Tall But  
ogo.html">Image Logo</a></  
" href="#">Tall  
Carousels</a>  
ith-slider.html">Variat  
ider.html">Testimoni
```

Array

Ordenación de un array

Método	Descripción
.reverse() ⚠	Invierte el orden de elementos del array.
.toReversed() ✓	Devuelve una copia del array, con el orden de los elementos invertido.
.sort() ⚠	Ordena los elementos del array bajo un criterio de ordenación alfabética .
.sort(criterio) ⚠	Idem, pero bajo un criterio de ordenación indicado por criterio.
.toSorted() ✓	Devuelve una copia del array, con los elementos ordenados.
.toSorted(criterio) ✓	Idem, pero ordenado por el criterio establecido por parámetro.

- ✓ El array original está seguro (no muta).
- ⚠ El array original cambia (muta).

Array

Método .reverse(Muta) y .toReversed(noMuta)

```
- □ ×  
  
const elements = ["A", "B", "C", "D", "E", "F"];  
  
const reversedElements = elements.reverse();  
  
reversedElements           // ["F", "E", "D", "C", "B", "A"]  
elements                  // ["F", "E", "D", "C", "B", "A"]  
reversedElements === elements // true
```

```
- □ ×  
  
const elements = ["A", "B", "C", "D", "E", "F"];  
  
const reversedElements = elements.toReversed();  
  
reversedElements           // ["F", "E", "D", "C", "B", "A"]  
elements                  // ["A", "B", "C", "D", "E", "F"]  
reversedElements === elements // false
```

Array

.structuredClone()

- □ ×

```
const elements = ["A", "B", "C", "D", "E", "F"];  
  
const reversedElements = structuredClone(elements).reverse();  
  
reversedElements           // ["F", "E", "D", "C", "B", "A"]  
elements                   // ['A', 'B', 'C', 'D', 'E', 'F']  
reversedElements === elements // false
```

Array

.sort() y .toSorted



```
const names = ["Alberto", "Zoe", "Ana", "Mauricio", "Bernardo"];  
  
const sortedNames = names.sort();  
  
sortedNames           // ["Alberto", "Ana", "Bernardo",  
"Mauricio", "Zoe"]  
names                 // ["Alberto", "Ana", "Bernardo",  
"Mauricio", "Zoe"]  
sortedNames === names // true
```

Array

.sort() y .toSorted

- □ ×

```
const names = ["Alberto", "Zoe", "Ana", "Mauricio", "Bernardo"];  
  
const sortedNames = names.toSorted();  
  
sortedNames           // ["Alberto", "Ana", "Bernardo", "Mauricio", "Zoe"]  
names                 // ["Alberto", "Zoe", "Ana", "Mauricio", "Bernardo"]  
sortedNames === names // false  
const numbers = [1, 8, 2, 32, 9, 7, 4];  
  
const sortedNumbers = numbers.toSorted();  
  
sortedNumbers    // [1, 2, 32, 4, 7, 8, 9]  
numbers          // [1, 8, 2, 32, 9, 7, 4]
```

Array

Función de comparación

- □ ×

```
const numbers = [1, 8, 2, 32, 9, 7, 4];

const alphabeticOrder = (a, b) => a + b;
const naturalOrder = (a, b) => a - b;

const alphaNumbers = numbers.toSorted(alphabeticOrder);      // [1, 8, 2, 32, 9, 7, 4]
const naturalNumbers = numbers.toSorted(naturalOrder);        // [1, 2, 4, 7, 8, 9, 32]
```

Array

Array Functions

Método	Descripción
UNDEFINED.forEach(f)	Ejecuta la función definida en f por cada uno de los elementos del array.
Comprobaciones	
BOOLEAN.every(f)	Comprueba si todos los elementos del array cumplen la condición de f.
BOOLEAN .some(f)	Comprueba si al menos un elemento del array cumple la condición de f.
Transformadores y filtros	
ARRAY.map(f)	Construye un array con lo que devuelve f por cada elemento del array.
ARRAY.filter(f)	Filtrá un array y se queda sólo con los elementos que cumplen la condición de f.
OBJECT.flat(level)	Aplana el array al nivel level indicado.
OBJECT.flatMap(f)	Aplana cada elemento del array, transformándolo según f. Equivale a .map().flat(1).

Array

Array functions

Método	Descripción
Búsquedas	
NUMBER.findIndex(f)	Devuelve la posición del elemento que cumple la condición de f.
OBJECT.find(f)	Devuelve el elemento que cumple la condición de f.
OBJECT.findLastIndex(f)	Idem a <code>findIndex()</code> , pero empezando a buscar desde el último elemento al primero.
OBJECT .findLast(f)	Idem a <code>find()</code> , pero empezando a buscar desde el último elemento al primero.
Acumuladores	
OBJECT.reduce(f, initial)	Ejecuta f con cada elemento (de izq a der), acumulando el resultado.
OBJECT .reduceRight(f, initial)	Idem al anterior, pero en orden de derecha a izquierda.

Array

.forEach

- □ ×

```
let numeros = [1, 2, 3, 4, 5];
```

// Utilizar forEach para iterar sobre cada elemento del array

```
numeros.forEach(function(elemento) {  
    console.log(elemento);  
});
```

Array

.every

- □ ×

```
let numerosPositivos = [1, 2, 3, 4, 5];

let todosPositivos = numerosPositivos.every(function(numero)
{
    return numero > 0;
});

console.log(todosPositivos); // Salida: true
```

Array

.some()

- □ ×

```
let numeros = [5, 8, 15, 3, 7];

let alMenosUnoMayorQue10 = numeros.some(function(numero) {
    return numero > 10;
});

console.log(alMenosUnoMayorQue10); // Salida: true
```

Array

.map()

- □ ×

```
const names = ["Ana", "PAblo", "pedro", "Angela"]
const nameSisas = names.map((names) => names.length);
console.log(nameSisas);
```

Array

.filter()

- □ ×

```
const names = ["Ana", "Pablo", "pedro", "Angela"]
const nameSisas =
names.filter((names) => names.startsWith("P"));
console.log(nameSisas); //Salida: [3, 6, 5, 7]
```

Array

.flatMap()

- □ ×

```
let numerosi = [1,2,,3,4,5]

let resultado = numerosi.flatMap(function(numero)
{
    return [numero,numero * 2]
});
console.log(resultado); //Salida: [1, 2, 2, 4, 3,
6,7, 8, 9, 10]
```

Array

.find() y .findIndex()

- □ ×

```
let usuarios = [
  { id: 1, nombre: 'Juan' },
  { id: 2, nombre: 'Ana' },
  { id: 3, nombre: 'Pedro' }
];

// Encontrar el usuario con ID igual a 2
let usuarioConId2 = usuarios.find(function(usuario)
{
  return usuario.id === 2;
});

console.log(usuarioConId2); // Salida: { id: 2,
                           nombre: 'Ana' }
```

Array

.findLast() y .findLastIndex()



```
let numeros = [5, 12, 8, 130, 44];

// Encontrar el último número mayor que 10 usando
// .reverse() y .find()
let ultimoNumeroMayorQue10 =
  [...numeros].reverse().find(function(numero) {
    return numero > 10;
});

console.log(ultimoNumeroMayorQue10); // Salida: 44
```

Array

.reduce()

- □ ×

```
let palabras = ['hola', 'mundo', 'hola', 'javascript', 'mundo', 'javascript', 'hola'];

// Utilizando reduce para contar la frecuencia de cada palabra
let frecuenciaPalabras = palabras.reduce(function(acumulador, palabra) {
    // Si la palabra ya está en el acumulador, incrementa su frecuencia
    if (palabra in acumulador) {
        acumulador[palabra]++;
    } else {
        // Si la palabra no está en el acumulador, agrega la palabra con frecuencia 1
        acumulador[palabra] = 1;
    }
    return acumulador;
}, {});

console.log(frecuenciaPalabras);
// Salida: { hola: 3, mundo: 2, javascript: 2 }
```

Array

.reduceRight

- □ ×

```
const numbers = [95, 5, 25, 10, 25];  
  
numbers.reduce((first, second) => first - second);  
// 95 - 5 - 25 - 10 - 25. Devuelve 30  
  
numbers.reduceRight((first, second) => first - second);  
// 25 - 10 - 25 - 5 - 95. Devuelve -110
```

Array

Restructuración básica

- □ ×

```
const elements = [5, 2];
const [first, last] = elements;    // first = 5, last =
2
```

```
const elements = [5, 4, 3, 2];
const [first, second] = elements; // first = 5, second
= 4, rest = discard
```

```
const elements = [5, 4, 3, 2];
const [first, , third] = elements; // first = 5, third
= 3, rest = discard
```

```
const elements = [4];
const [first, second] = elements; // first = 4, second
= undefined
```

Array

Intercambio de variables

- □ ×

```
let a = 5;
let b = 10;

// Intercambio de valores utilizando la desestructuración de arrays
[a, b] = [b, a];

console.log("Después del intercambio:");
console.log("a =", a); // Salida: 10
console.log("b =", b); // Salida: 5
```

Array

Spread (Extraer y Recoger)

- □ ×

```
let frutas = ['manzana', 'plátano', 'uva', 'naranja'];

// Extraer los dos primeros elementos y recoger el resto en una
variable
let [primeraFruta, segundaFruta, ...restoFrutas] = frutas;

console.log(primerFruta); // Salida: 'manzana'
console.log(segundaFruta); // Salida: 'plátano'
console.log(restoFrutas); // Salida: ['uva', 'naranja']
```

Array

Spread (Intercambiar Elementos)

- □ ×

```
let a = 5;
let b = 10;

// Intercambiar valores utilizando desestructuración con spread
[a, b] = [b, a];

console.log("Después del intercambio:");
console.log("a =", a); // Salida: 10
console.log("b =", b); // Salida: 5
```

Array

Spread (en Funciones)



```
function obtenerCoordenadas() {  
    return [42.876, -8.544];  
}  
  
// Desestructuración con spread al llamar a la función  
let [latitud, longitud] = obtenerCoordenadas();  
  
console.log("Latitud:", latitud); // Salida: 42.876  
console.log("Longitud:", longitud); // Salida: -8.544
```

Array

Rest (Agrupar) Debug

```
const debug = (...param) => console.log(param);
debug(1, 2, 3, 4, 5);

// [1, 2, 3, 4, 5]
```

</Be a
coder>