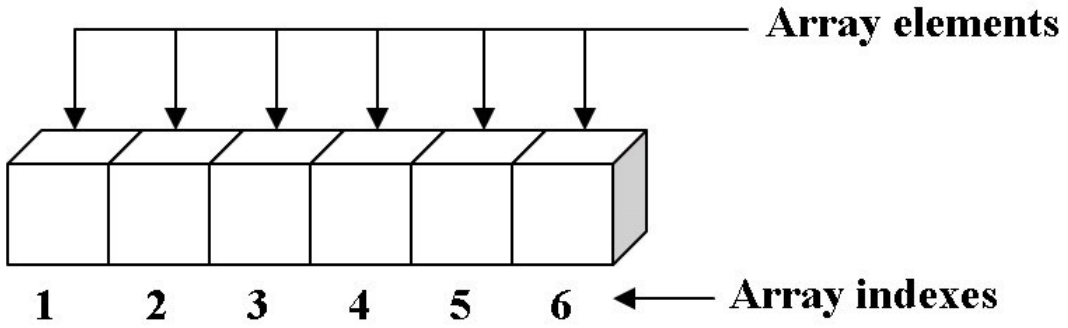
ARRAYS EN JAVASCRIPT

Charly Cimino

Arrays en JavaScript

Charly Cimino

Versión 2023.08.07



One-dimensional array with six elements

Los arrays en JS son dinámicos y mutables. El lenguaje los trata como objetos, incluyendo métodos muy útiles.

Referencia completa: https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/from

Crear un array literalmente

```
const el_array = [item1, item2, ...];
```

```
const colores = ["Rojo", "Azul", "Lila", "Gris"];
```

Un array de 4 cadenas.

```
const numeros = [14, 0, -5];
```

Un array de 3 enteros.

```
const personas = [
    { nombre: "Luis", edad: 27 },
    { nombre: "Ana", edad: 22 },
    { nombre: "Pepe", edad: 33 },
    { nombre: "Maria", edad: 25 }
];
```

Un array de 3 objetos.
Una declaración puede hacerse ocupando más de una línea.

```
const animales = [];
animales[0] = "Búho";
animales[1] = "Gato";
animales[2] = "Perro";
```

Un array de 3 cadenas.
Puede partirse de un array vacío y luego agregar elementos.

https://www.w3schools.com/js/js_arrays.asp

Crear un array vía constructor/métodos

```
const colores = new Array("Rojo", "Azul", "Lila");
console.log(colores); // ["Rojo", "Azul", "Lila"]
```

```
const colores = Array.of("Rojo", "Azul", "Lila");
console.log(colores); // ["Rojo", "Azul", "Lila"]
```

El constructor de Array y el método estático 'of' producen el mismo resultado cuando tienen más de un parámetro.

```
const colores = new Array(4);
console.log(colores); // [ , , , ]
```

```
const colores = Array.of(4);
console.log(colores); // [4]
```

El constructor de Array y el método estático 'of' producen distinto resultado cuando tienen un solo parámetro entero.

```
const letras = Array.from("Hola");
console.log(letras); // ["H", "o", "l", "a"]

const cjtoPares = new Set();
cjtoPares.add(0);
cjtoPares.add(2);
cjtoPares.add(4);
cjtoPares.add(6);
const arrPares = Array.from(cjtoPares);
console.log(arrPares); // [0, 2, 4, 6]
```

El método estático 'from' permite crear un array a partir de los valores de un iterable.

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/of

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/from

Operaciones básicas

```
const colores = ["Rojo", "Azul", "Lila"];
console.log(colores); // ["Rojo", "Azul", "Lila"]
```

Mostrar un array directamente.

```
const colores = ["Rojo", "Azul", "Lila"];
console.log(colores.length); // 3
```

Obtener la longitud.

```
const colores = ["Rojo", "Azul", "Lila"];
const elAzul = colores[1];
console.log(elAzul); // "Azul"
```

Obtener elemento en determinado índice.

```
const colores = ["Rojo", "Azul", "Lila"];
colores[1] = "Gris";
console.log(colores); // ["Rojo", "Gris", "Lila"]
```

Establecer elemento en determinado índice.

```
const colores = ["Rojo", "Azul", "Lila"];
colores[-1] = "Gris";
console.log(colores); // ["Rojo", "Gris", "Lila"]
console.log(colores.length); // 3
console.log(colores[-1]); // "Gris"
```

Índices negativos: permitidos pero no son parte del array como tales.

https://stackoverflow.com/questions/13618571/should-negative-indexes-in-javascript-arrays-contribute-to-array-length

```
const colores = ["Rojo", "Azul"];
colores[7] = "Gris";
console.log(colores); // ["Rojo", "Azul", , , , , , "Gris"]
console.log(colores.length); // 8
console.log(colores[4]); // undefined
```

Array con "huecos".

https://blog.caplin.com/2012/01/13/javascript-is-hard-part-1-you-cant-trust-arrays/

https://www.w3schools.com/js/js_array_methods.asp

Agregar / Quitar

```
const colores = ["Rojo", "Azul"];
colores.push("Lila");
console.log(colores); // ["Rojo", "Azul", "Lila"]
const tam = colores.push("Oro");
console.log(colores); // ["Rojo", "Azul", "Lila", "Oro"]
console.log(tam); // 4
```

Agregar elemento al final (retorna nueva longitud).

```
const colores = ["Rojo", "Azul"];
colores.unshift("Lila");
console.log(colores); // ["Lila", "Rojo", "Azul"]
const tam = colores.unshift("Gris");
console.log(colores); // ["Oro", "Lila", "Rojo", "Azul"]
console.log(tam); // 4
```

Agregar elemento al principio (retorna nueva longitud).

```
const colores = ["Rojo", "Azul", "Lila"];
const ultimo = colores.pop(); // "Lila"
console.log(colores); // ["Rojo", "Azul"]
```

Eliminar elemento al final (retorna elemento eliminado).

```
const colores = ["Rojo", "Azul", "Lila"];
const primero = colores.shift(); // "Rojo"
console.log(colores); // ["Azul", "Lila"]
```

Eliminar elemento al principio (retorna elemento eliminado).

```
const colores = ["Rojo", "Azul"];
// arr.splice(inicio, cantBorrar, item1, item2, ...)
const borrados = colores.splice(2, 0, "Lila", "Gris");
console.log(colores); // ["Rojo", "Azul", "Lila", "Gris"]
console.log(borrados); // []
```

Agregar elementos con 'splice'

```
const colores = ["Rojo", "Azul", "Lila", "Gris"];
// arr.splice(inicio, cantBorrar, item1, item2, ...)
const borrados = colores.splice(1, 2);
console.log(colores); // ["Rojo", "Gris"]
console.log(borrados); // ["Azul", "Lila"]
```

Eliminar elementos con 'splice'

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/push
https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/pop
https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/splice

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/shift
https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/unshift
https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/splice

Quitar con 'delete'

```
const colores = ["Rojo", "Azul", "Lila", "Gris"];
delete colores[2]; // Similar a: colores[2] = undefined
console.log(colores); // ["Rojo", "Azul", undefined, "Gris"]
console.log(colores[2]); // undefined
```

Deja "huecos" en el array.

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/delete

Obtener arrays a partir de otros

Estos métodos no modifican el array sobre el que se invocan. Devuelven uno nuevo.

```
const frios = ["Azul", "Morado"];
const calidos = ["Rojo", "Naranja"];
const monos = ["Blanco", "Gris", "Negro"];

const friosYCalidos = frios.concat(calidos);
const calidosYFrios = calidos.concat(frios);

const todos = monos.concat(frios, calidos);

console.log(friosYCalidos); // ["Azul", "Morado", "Rojo", "Naranja"]
console.log(calidosYFrios); // ["Rojo", "Naranja", "Azul", "Morado"]
console.log(todos); // ["Blanco", "Gris", "Negro", "Azul", "Morado", "Rojo", "Naranja"]
```

Obtener la concatenación de dos o más arrays.

```
const colores = ["Blanco", "Gris", "Negro", "Azul", "Morado", "Rojo", "Naranja"];

console.log(colores.slice(0,3)); // ["Blanco", "Gris", "Negro"]
console.log(colores.slice(5,colores.length)); // ["Rojo", "Naranja"]
console.log(colores.slice(2,0)); // []
const arrCopia = colores.slice(); // Nuevo array con los mismos elementos (array copia)
```

Obtener una porción de un array.

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/concat

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/slice

Obtener arrays a partir de otros

El método `flat` permite "planchar" un array de arrays hasta la profundidad especificada como parámetro (opcional, por defecto es **1**)

```
let nuevoArray = arr.flat(profundidad)
```
(Opc.)

```
let arr1 = [1, 2, [3, 4]];
console.log(arr1.flat()); // [1, 2, 3, 4]

let arr2 = [1, 2, [3, 4, [5, 6]]];
console.log(arr2.flat()); // [1, 2, 3, 4, [5, 6]]

let arr3 = [1, 2, [3, 4, [5, 6]]];
console.log(arr3.flat(2)); // [1, 2, 3, 4, 5, 6]
```

No modifica el array sobre el que se invoca. Devuelve uno nuevo.

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/flat

Obtener cadenas a partir de arrays

```
const colores = ["Rojo", "Azul", "Lila", "Gris"];
const coloresStr = colores.toString()
console.log(colores); // ["Rojo", "Azul", "Lila", "Gris"]
console.log(coloresStr); // "Rojo,Azul,Lila,Gris"
```

Obtener representación del array como cadena.

```
const colores = ["Rojo", "Azul", "Lila", "Gris"];

console.log(colores.join()); // "Rojo,Azul,Lila,Gris"
console.log(colores.join("")); // "RojoAzulLilaGris"
console.log(colores.join(" ")); // "Rojo Azul Lila Gris"
console.log(colores.join("-")); // "Rojo-Azul-Lila-Gris"
console.log(colores.join(" || ")); // "Rojo || Azul || Lila || Gris"
```

Obtener valores del array en una cadena con separador (opcional).

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/toString

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/join

Modificar arrays

Estos métodos modifican el array sobre el que se invocan. Devuelven el mismo array sobre el cual actuaron.

```
const colores = ["Rojo", "Azul", "Lila", "Gris"];
const invertidos = colores.reverse();

console.log(colores); // ["Gris", "Lila", "Azul", "Rojo"]
console.log(invertidos); // ["Gris", "Lila", "Azul", "Rojo"]
```

Invertir elementos de un array.

```
const colores = ["Rojo", "Azul", "Lila", "Gris"];

const primeros2Negros = colores.fill("Negro", 0, 2);
console.log(primeros2Negros); // ["Negro", "Negro", "Lila", "Gris"]
console.log(colores); // ["Negro", "Negro", "Lila", "Gris"]

const todosNegros = colores.fill("Negro");
console.log(todosNegros); // ["Negro", "Negro", "Negro", "Negro"]
console.log(colores); // ["Negro", "Negro", "Negro", "Negro"]
```

Rellenar valores en un array.

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/reverse

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/fill

Consultar en arrays

```
const colores = ["Rojo", "Azul", "Lila", "Gris"];

console.log(colores.includes("Azul")); // true
console.log(colores.includes("azul")); // false
```

Verificar existencia de un elemento en un array

```
const colores = ["Rojo", "Azul", "Lila", "Azul", "Gris"];

console.log(colores.indexOf("Azul")); // 1
console.log(colores.indexOf("azul")); // -1
```

Índice de la primera ocurrencia del elemento en el array (-1 si no existe)

```
const colores = ["Rojo", "Azul", "Lila", "Azul", "Gris"];

console.log(colores.lastIndexOf("Azul")); // 3
console.log(colores.lastIndexOf("azul")); // -1
```

Índice de la última ocurrencia del elemento en el array (-1 si no existe)

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/includes

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/indexOf
https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/lastIndexOf

Iterar elementos de un array

Imperativo

```
const colores = ["Rojo", "Azul", "Lila", "Gris"];

for (let i = 0; i < colores.length; i++) {
    console.log(colores[i]);
}
```

Mostrar elementos con for

```
const colores = ["Rojo", "Azul", "Lila", "Gris"];

for (const color of colores) {
    console.log(color);
}
```

Mostrar elementos con for...of

Declarativo

```
const colores = ["Rojo", "Azul", "Lila", "Gris"];

function mostrar(x) {
    console.log(x);
}

colores.forEach(mostrar);
```

Mostrar elementos con foreach (función de orden superior)

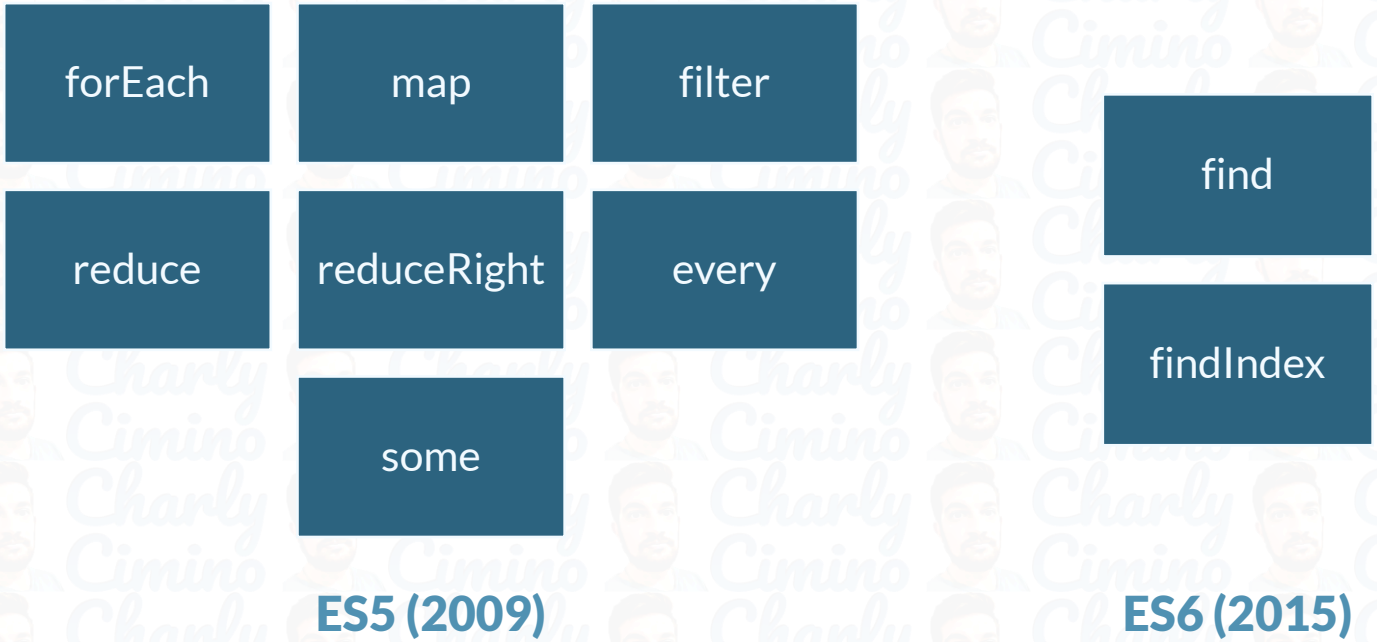
```
const colores = ["Rojo", "Azul", "Lila", "Gris"];

colores.forEach(color => {
    console.log(color);
});
```

Mostrar elementos con foreach y función flecha anónima.

Funciones de orden superior en JS

Permiten iterar arrays de manera funcional, con una sintaxis declarativa y limpia.



foreach

Itera el array aplicando una función a cada elemento, retornando **undefined**.

```
let nuevoArray = arr.foreach(function callback(valorActual, indice, array) { // Operaciones })
```

(Opc.) (Opc.)

```
const colores = ["Rojo", "Azul", "Lila", "Gris"];
let cad = "";

colores.forEach(color => {
    cad += color + " | ";
});

console.log(cad); // "Rojo | Azul | Lila | Gris |"
```

```
const colores = ["Rojo", "Azul", "Lila", "Gris"];
let cad = "";

colores.forEach((color, idx, colores) => {
    cad += (idx+1) + ") " + color;
    cad += ((idx+1) < colores.length ? " | " : "");
});

console.log(cad); // "1) Rojo | 2) Azul | 3) Lila | 4) Gris"
```

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/forEach

map

Itera el array aplicando una función a cada elemento, retornando un nuevo array con los resultados.

```
let nuevoArray = arr.map(function callback(valorActual, indice, array) { // Nuevo elemento })
```

(Opc.) (Opc.)

```
const colores = ["Rojo", "Azul", "Lila", "Gris"];

const coloresUpper = colores.map(elem => elem.toUpperCase());

console.log(coloresUpper); // ["ROJO", "AZUL", "LILA", "GRIS"]
```

```
const numeros = [1, 5, 10, 15];

const dobles = numeros.map(n => n * 2);

console.log(dobles); // [2, 10, 20, 30]
```

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/map

filter

Itera el array aplicando un predicado a cada elemento, retornando un nuevo array con aquellos que hayan sido **`true`**.

```
let nuevoArray = arr.filter(function callback(valorActual, indice, array) { // Predicado })
```

(Opc.) (Opc.)

```
const colores = ["Rojo", "Azul", "Negro", "Gris", "Verde"];

const coloresMasDe4 = colores.filter(elem => elem.length > 4);

console.log(coloresMasDe4 ); // ["Negro", "Verde"]
```

```
const numeros = [1, 4, 11, 16];

const pares = numeros.filter(n => n % 2 == 0);

console.log(pares); // [4, 16]
```

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/filter

reduce

Itera el array aplicando una función reductora a cada elemento, retornando un único valor como resultado.

```
let valor = arr.reduce(function callback(acumulado, valorActual, indice, array) { // Reducción }, valorInicial)
```

(Opc.) (Opc.) (Opc.)

```
const numeros = [1, 2, 3, 4];

const suma = numeros.reduce((acu, n) => acu + n);

console.log(suma); // 10
```

```
const numeros = [1, 2, 3, 4];

const resta = numeros.reduce((acu, n) => acu - n);

console.log(resta); // -8
```

```
const numeros = [1, 2, 3, 4];

const suma = numeros.reduce(((acu, n) => acu + n), 10);

console.log(suma); // 20
```

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/Reduce

reduceRight

Itera el array aplicando una función reductora a cada elemento comenzando desde la derecha, retornando un único valor como resultado.

```
let valor = arr.reduceRight(function callback(acumulado, valorActual, indice, array) { // Reducción }, valorInicial)
                                                                        (Opc.)  (Opc.)          (Opc.)
```

```
const numeros = [1, 2, 3, 4];

const suma = numeros.reduceRight((acu, n) => acu + n);

console.log(suma); // 10
```

```
const numeros = [1, 2, 3, 4];

const resta = numeros.reduceRight((acu, n) => acu - n);

console.log(resta); // -2
```

```
const numeros = [1, 2, 3, 4];

const suma = numeros.reduceRight(((acu, n) => acu + n), 10);

console.log(suma); // 20
```

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/ReduceRight

every

Itera el array aplicando un predicado a cada elemento, retornando `true` si **todos** los resultados dan `true`.

```
let todosCumplen = arr.every(function callback(valorActual, indice, array) { // Predicado })
```

(Opc.) (Opc.)

```
const numeros = [1, 2, 3, 4];

const todosPositivos = numeros.every(n => n > 0);
const todosPares = numeros.every(n => n % 2 == 0);

console.log(todosPositivos); // true
console.log(todosPares); //false
```

¡Llamar este método en un array vacío devuelve true para cualquier condición!

```
const numeros = [];

const deberiaSerFalse = numeros.every(n => false);

console.log(deberiaSerFalse); // true
```

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/every

some

Itera el array aplicando un predicado a cada elemento, retornando `true` si **algún** resultado da `true`.

```
let algunoCumple = arr.some(function callback(valorActual, indice, array) { // Predicado })
```

(Opc.) (Opc.)

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

const algunNegativo = numeros.some(n => n < 0);
const algunPar = numeros.some(n => n % 2 == 0);

console.log(algunNegativo); // false
console.log(algunPar); // true
```

¡Llamar este método en un array vacío devuelve false para cualquier condición!

```
const numeros = [];

const deberiaSerTrue = numeros.some(n => true);

console.log(deberiaSerTrue); // false
```

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/some

find

Itera el array en busca de un elemento, retornando el valor de la primera ocurrencia o **`undefined`** si no se encuentra.

```
let elemento = arr.find(function callback(elementoABuscar, indice, array) { // Predicado })
```

(Opc.) (Opc.)

```
const personas = [{
    nombre: "Luis", edad: 27
}, {
    nombre: "Ana", edad: 22
}, {
    nombre: "Pepe", edad: 33
}, {
    nombre: "Maria", edad: 25
}];

const pepe = personas.find(p => p.nombre === "Pepe");
console.log(pepe); // {nombre: "Pepe", edad: 33}
```

```
const personas = [{
    nombre: "Luis", edad: 27
}, {
    nombre: "Ana", edad: 22
}, {
    nombre: "Pepe", edad: 33
}, {
    nombre: "Maria", edad: 25
}];

const juan = personas.find(p => p.nombre === "Juan");
console.log(juan); // undefined
```

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/find

findIndex

Itera el array en busca de un elemento, retornando el índice de la primera ocurrencia o **-1** si no se encuentra.

```
let iElemento = arr.findIndex(function callback(elementoABuscar, indice, array) { // Predicado })
```
(Opc.) (Opc.)

```
const personas = [{
    nombre: "Luis", edad: 27
}, {
    nombre: "Ana", edad: 22
}, {
    nombre: "Pepe", edad: 33
}, {
    nombre: "Maria", edad: 25
}];

const idx = personas.findIndex(p => p.nombre === "Pepe");
console.log(idx); // 2
```

```
const personas = [{
    nombre: "Luis", edad: 27
}, {
    nombre: "Ana", edad: 22
}, {
    nombre: "Pepe", edad: 33
}, {
    nombre: "Maria", edad: 25
}];

const idx = personas.findIndex(p => p.nombre === "Juan");
console.log(idx); // -1
```

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/findIndex

Ordenar elementos de un array

El método `sort` modifica el array sobre el que se invoca. Devuelve el mismo array sobre el cual actuó.

```
const colores = ["Rojo", "Azul", "Lila", "Gris"];

console.log(colores.sort()); // ["Azul", "Gris", "Lila", "Rojo"]
console.log(colores.reverse()); // ["Rojo", "Lila", "Gris", "Azul"]
```

Las cadenas se ordenan por defecto alfabéticamente.

```
const numeros = [40, 100, 1, 5, 25, 10];

console.log(numeros.sort()); // [1, 10, 100, 25, 40, 5]
```

Los números se ordenan como si fueran cadenas, provocando resultados incoherentes.

Solución: Proveer al sort un método de comparación

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/sort

Ordenar elementos de un array

El método `sort` puede recibir como parámetro (opcional) una función de comparación.

```
const numeros = [40, 100, 1, 5, 25, 10];

numeros.sort((a, b) => a - b);
console.log(numeros); // [1, 5, 10, 25, 40, 100]
```

Números ordenados de forma ascendente.

```
const numeros = [40, 100, 1, 5, 25, 10];

numeros.sort((a, b) => b - a);
console.log(numeros); // [100, 40, 25, 10, 5, 1]
```

Números ordenados de forma descendente.

```
const personas = [
        { nombre: "Luis", edad: 27 },
        { nombre: "Ana", edad: 22 },
        { nombre: "Pepe", edad: 33 },
        { nombre: "Maria", edad: 25 }
    ];

personas.sort((p1, p2) => {
    return (p1.nombre > p2.nombre ? 1 : -1);
});
/*
    0: {nombre: "Ana", edad: 22}
    1: {nombre: "Luis", edad: 27}
    2: {nombre: "Maria", edad: 25}
*/
```

Personas ordenadas por nombre de forma ascendente.

```
const personas = [
        { nombre: "Luis", edad: 27 },
        { nombre: "Ana", edad: 22 },
        { nombre: "Maria", edad: 25 }
    ];

personas.sort((p1, p2) => p2.edad - p1.edad);
/*
    0: {nombre: "Luis", edad: 27}
    1: {nombre: "Maria", edad: 25}
    2: {nombre: "Ana", edad: 22}
*/
```

Personas ordenadas por edad de forma descendente.

FIN DE LA PRESENTACIÓN

Encontrá más como estas en mi sitio web.