

Desarrollo Web en Entorno Cliente

Tema 6





[arrays]

Los arrays, también llamados matrices, vectores o arreglos, son un tipo de datos no primitivo; realmente son objetos, similares a una lista indexada.

JS proporciona métodos para realizar operaciones útiles, como recorrerlos o cambiar su contenido.

Tanto la longitud como el tipo de los elementos de un array no son fijos y pueden cambiar a lo largo del código.

Este hecho hace que los datos del array se puedan almacenar en ubicaciones no contiguas de memoria, lo que contrasta con el uso y funcionamiento de los arrays en otro tipo de lenguajes de programación, como, por ejemplo, C.

Sirven para agrupar elementos relacionados entre sí.

Los arrays se crean como cualquier otra variable o constante, pero al inicializarla se colocan corchetes `[]`, con los elementos del array entre ellos y separados por comas `,`.

```
const miArray = ["Hola", "Jorge", 1]);  
console.log(typeof miArray);
```

object

[arrays]

Existe otra forma de crear arrays, mediante la palabra reservada *new* junto con el constructor del objeto *Array* seguido de los argumentos entre paréntesis que, en este caso, serán los elementos que va a contener el array:

```
const miArray = new Array("Hola", "Jorge", 1);  
  
console.log(miArray);
```

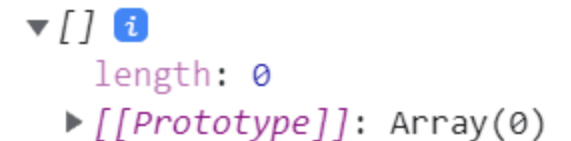


```
▼ Array(3) ⓘ  
  0: "Hola"  
  1: "Jorge"  
  2: 1  
  length: 3  
  ► [[Prototype]]: Array(0)
```

También se pueden crear arrays vacíos:

```
const miArray = new Array();  
  
console.log(miArray);
```

```
const miArray = [];  
  
console.log(miArray);
```



```
▼ [] ⓘ  
  length: 0  
  ► [[Prototype]]: Array(0)
```

[arrays]

Los arrays pueden contener todo tipo de datos en su interior, esto también es una característica diferente con respecto a la mayoría de otros lenguajes de programación.

Pueden contener números, strings, arrays, booleanos... incluso objetos y otros arrays.

```
const miArray = new Array("Hola", "Jorge", 1,
true, null,
{marca: "Seat", modelo: "León", mostrarMarca:
function () {return this.marca}});
```

```
console.log(miArray);
console.log(miArray[5].mostrarMarca());
```

```
const miArray2 = [[1,2], "Jorge", 1];
```

```
console.log(miArray2);
```

[arrays]

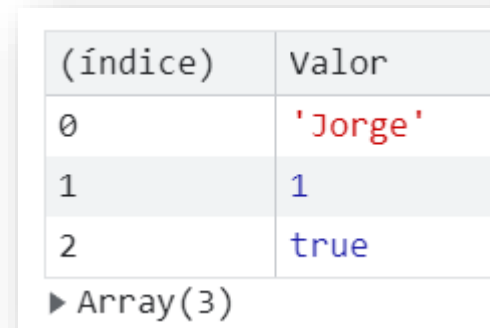
Los arrays utilizan índices, números naturales, para ordenar y acceder a los elementos del array, se le conoce como la posición del elemento en el array.

El primer elemento de un array comienza en el índice 0 (no en 1).

Es muy útil `console.table` para mostrar los elementos de un array.

Para acceder a un elemento determinado del array se pone el nombre del array seguido de corchetes `[]` y entre ellos el índice donde está dicho elemento.

```
const miArray = ["Jorge", 1, true];  
console.table (miArray);
```



(índice)	Valor
0	'Jorge'
1	1
2	true

► Array(3)

```
console.log (miArray[0]);
```

Jorge

[arrays]

Si se trata de acceder a un elemento que no existe se obtiene como resultado *undefined*.

```
const meses=["Enero","Febrero","Marzo","Abril"];  
console.log(meses[4]);
```

undefined

Para acceder a una posición de un array contenido dentro de otro array, se necesitan dos índices:

```
const meses=["Enero","Febrero","Marzo","Abril",  
["1ª Quincena de Mayo", "2ªQuincena de Mayo"]];  
console.log(meses[4][0]);
```

1ª Quincena de Mayo

[modificar elementos]

Los arrays se pueden alterar, modificando los valores que contienen en las distintas posiciones.

A pesar de que un array está declarado como constante sus valores pueden modificarse.

Los únicos casos en los que los valores de una variable declarada como constante se pueden modificar es en los tipos arrays, funciones anónimas y objetos (siempre que el objeto no esté congelado).

```
const meses=["Enero","Febrero","Marzo","Abril"];

meses[0]="Diciembre";
console.log(meses);
```

```
▼ (4) ['Diciembre', 'Febrero', 'Marzo', 'Abril']
  0: "Diciembre"
  1: "Febrero"
  2: "Marzo"
  3: "Abril"
  length: 4
  ► [[Prototype]]: Array(0)
```


[modificar elementos]

De igual manera, se pueden agregar más elementos al array accediendo a sus posiciones y asignándoles valores.

```
const meses=["Enero","Febrero",  
"Marzo","Abril"];  
  
meses[4]="Mayo";  
meses[6]="Junio";  
  
console.log(meses);
```

```
▼ (7) ['Enero', 'Febrero', 'Marzo', 'Abril', 'Mayo', vacío, 'Junio']  
  0: "Enero"  
  1: "Febrero"  
  2: "Marzo"  
  3: "Abril"  
  4: "Mayo"  
  6: "Junio"  
  length: 7  
  ► [[Prototype]]: Array(0)
```

[push]

Hay disponibles métodos de gran utilidad proporcionados por el objeto Array, son conocidos como los *Array Methods*.

Uno de ellos, *push()*, que añade al final del array el elemento que recibe como argumento .

```
const meses=["Enero","Febrero",  
"Marzo","Abril"];
```

```
meses.push("Mayo");  
meses.push("Junio");
```

```
console.log(meses);
```

```
▼ (6) ['Enero', 'Febrero', 'Marzo', 'Abril', 'Mayo', 'Junio']  
  0: "Enero"  
  1: "Febrero"  
  2: "Marzo"  
  3: "Abril"  
  4: "Mayo"  
  5: "Junio"  
  length: 6  
  ► [[Prototype]]: Array(0)
```

(unshift)

Otro de los métodos del objeto *Array* (*Array Methods*) es *unshift()*.

unshift(), a diferencia de *push()*, permite añadir el elemento al comienzo del array

```
const meses=["Enero","Febrero",  
"Marzo","Abril"];  
  
meses.unshift("Diciembre");  
  
console.log(meses);
```

```
▼ (5) ['Diciembre', 'Enero', 'Febrero', 'Marzo', 'Abril']  
  0: "Diciembre"  
  1: "Enero"  
  2: "Febrero"  
  3: "Marzo"  
  4: "Abril"  
  length: 5  
  ► [[Prototype]]: Array(0)
```

(pop)

El método *pop()*, “saca” el último elemento del array, el elemento que tiene el índice mayor.

Se puede recoger o almacenar, no es obligatorio, el valor contenido en el array que se acaba de sacar.

```
const meses=["Enero","Febrero",  
"Marzo","Abril"];
```

```
const mes=meses.pop();
```

```
console.log(meses);
```

```
▼ (3) ['Enero', 'Febrero', 'Marzo']  
  0: "Enero"  
  1: "Febrero"  
  2: "Marzo"  
  length: 3  
  ► [[Prototype]]: Array(0)
```

[shift]

Y, de forma parecida, se puede “sacar” elementos por el principio del array con el método *shift()*.

```
const meses=["Enero","Febrero",  
"Marzo","Abril"];  
  
const mes=meses.shift();  
  
console.log(mes);  
console.log(meses);
```



The screenshot shows a variable named 'Enero' in the browser's developer console. It displays an array with three elements: 'Febrero', 'Marzo', and 'Abril'. The array is represented as (3) ['Febrero', 'Marzo', 'Abril']. The indices are shown as 0: 'Febrero', 1: 'Marzo', and 2: 'Abril'. The length of the array is 3. The prototype is shown as [[Prototype]]: Array(0).

```
Enero  
▼ (3) ['Febrero', 'Marzo', 'Abril']  
  0: "Febrero"  
  1: "Marzo"  
  2: "Abril"  
  length: 3  
  ► [[Prototype]]: Array(0)
```

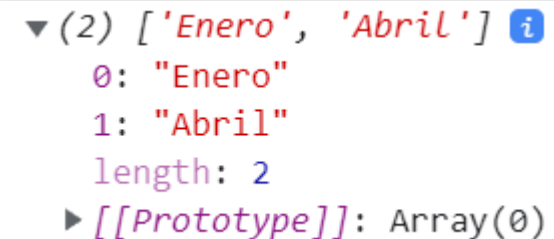
[splice]

`splice()` es otro método útil, que sirve para eliminar o agregar elementos a un array.

Si recibe únicamente dos argumentos, el primero es la posición desde donde se quiere eliminar los elementos y el segundo es el número de elementos que se quieren eliminar, recolocando los valores y los índices del array.

Devuelve un array con los elementos que ha eliminado.

```
const meses=["Enero","Febrero",  
"Marzo","Abril"];  
  
meses.splice(1,2);  
  
console.log(meses);
```



```
▼ (2) ['Enero', 'Abril'] ⓘ  
  0: "Enero"  
  1: "Abril"  
  length: 2  
  ► [[Prototype]]: Array(0)
```

[splice]

Si `splice()` recibe más de dos argumentos, el tercer argumento y siguientes se agregarán al array, empezando en el índice especificado por el primer argumento.

```
const meses=["Enero","Febrero",  
"Marzo","Abril"];  
  
meses.splice(1,0,"Junio","Julio");  
  
console.log(meses);
```

```
const meses=["Enero","Febrero",  
"Marzo","Abril"];  
  
meses.splice(1,2,"Junio","Julio");  
  
console.log(meses);
```

```
▼ (6) ['Enero', 'Junio', 'Julio', 'Febrero', 'Marzo', 'Abril']  
  0: "Enero"  
  1: "Junio"  
  2: "Julio"  
  3: "Febrero"  
  4: "Marzo"  
  5: "Abril"  
  length: 6  
  ► [[Prototype]]: Array(0)
```

```
▼ (4) ['Enero', 'Junio', 'Julio', 'Abril']  
  0: "Enero"  
  1: "Junio"  
  2: "Julio"  
  3: "Abril"  
  length: 4  
  ► [[Prototype]]: Array(0)
```

[slice]

`slice()` es un método para obtener parte de los elementos de un array, una “rebanada” de elementos del array.

Recibe dos argumentos, el primero es el índice del elemento que se quiere extraer, si se omite o es un valor negativo se considerará desde la posición 0, el segundo es el índice del elemento hasta el que se quiere obtener (sin incluirlo), si se omite o es mayor que los elementos del array se considera hasta el último elemento (incluido), si es negativo se considera 0.

Devuelve un array con los elementos que cumplen el rango de parámetros, el array original permanece inalterado.

```
const meses=["Enero", "Febrero",  
"Marzo", "Abril"];  
  
console.log(meses.slice(1,3));
```

```
▼ (2) ['Febrero', 'Marzo']  
  0: "Febrero"  
  1: "Marzo"  
  length: 2  
  ► [[Prototype]]: Array(0)
```


[reverse y toReversed]

`reverse()` es un método que reubica al revés los elementos del array, le da la vuelta al array, pasando a ser el último elemento el primero, el penúltimo pasa a ser el segundo,... hasta el primer elemento que pasa a ser el último, devuelve el array modificado.

El array original queda modificado. Si no se quiere modificar el array debe utilizarse el método `toReversed()` que devuelve un array al revés, pero no modifica el array original.

```
const meses=["Enero","Febrero",  
"Marzo","Abril"];  
  
console.log(meses.toReversed());  
  
console.log(meses.reverse());  
  
console.log(meses);
```

```
► (4) ['Abril', 'Marzo', 'Febrero', 'Enero']  
► (4) ['Abril', 'Marzo', 'Febrero', 'Enero']  
► (4) ['Abril', 'Marzo', 'Febrero', 'Enero']
```

[concat]

Se pueden crear nuevos arrays a partir de los elementos de otros arrays con el método *concat()*.

Este método incorpora los elementos de un array a continuación de los suyos propios, pero el array permanece inalterado, en su lugar, devuelve un array con la suma de los dos.

```
const meses=["Enero","Febrero",  
"Marzo","Abril"];  
  
const otrosMeses=["Mayo","Junio"];  
  
const año=meses.concat(otrosMeses);  
  
console.log(meses);  
console.log(año);
```

```
► (4) ['Enero', 'Febrero', 'Marzo', 'Abril']  
► (6) ['Enero', 'Febrero', 'Marzo', 'Abril', 'Mayo', 'Junio']
```

[join]

`join()` es un método que crea y devuelve una nueva cadena de caracteres concatenando todos los elementos del array, separados por comas o una cadena de separación específica.

Recibe como argumento la cadena de separación que se desea utilizar para separar cada par de elementos del array. Si se omite, los elementos se separan con una coma (",").

Si el array solo tiene un elemento se devuelve sin usar el separador.

```
const meses=["Enero","Febrero","Marzo","Abril"];  
console.log(meses.join());
```

Enero,Febrero,Marzo,Abril

[copiar arrays]

También se puede utilizar el *Spread Operator* con arrays, de forma análoga a como se realiza con objetos, para crear nuevos arrays que contengan elementos de otros arrays.

```
const meses=["Enero","Febrero",  
"Marzo","Abril"];  
  
const otrosMeses=["Mayo","Junio"];  
  
const año=[...meses, ...otrosMeses];  
  
console.log(año);
```

```
▼ (6) ['Enero', 'Febrero', 'Marzo', 'Abril', 'Mayo', 'Junio']  
  0: "Enero"  
  1: "Febrero"  
  2: "Marzo"  
  3: "Abril"  
  4: "Mayo"  
  5: "Junio"  
  length: 6  
  ► [[Prototype]]: Array(0)
```

[destructuring arrays]

También se puede hacer desestructuración con arrays.

En los arrays se realiza con los corchetes `[]` en lugar de las llaves `{}`. Entre los corchetes y separadas por comas `,` se pone el nombre de la variable o constante se deseen; se van a ir recuperando los elementos y asignando a las variables en orden, por el índice, y de uno en uno.

```
const meses=["Enero","Febrero",  
"Marzo","Abril"];
```

```
const [mes1, mes2]=meses;
```

```
console.log(mes1);
```

```
console.log(mes2);
```

Enero
Febrero

[destructuring arrays]

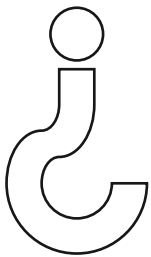
Para acceder al tercer elemento del array usando la desestructuración, sin querer recoger los dos primeros, se ponen comas, pero sin ningún nombre de variable o constante.

```
const meses=["Enero","Febrero",  
"Marzo","Abril"];
```

```
const [, , mes3]=meses;
```

```
console.log(mes3);
```

Marzo



```
const {3:mes3}=meses;
```

```
console.log(mes3);
```

Abril



[destructuring arrays]

Se puede usar el *Spread Operator* con la desestructuración para acceder a unos elementos y el resto de los elementos del array almacenarlos en un array nuevo.

```
const meses=["Enero","Febrero",  
"Marzo","Abril","Mayo","Junio",  
"Julio","Agosto"];  
  
const [, ,mes3,mes4, ...restoMeses]=meses;  
  
console.log(mes3);  
console.log(mes4);  
console.log(restoMeses);
```

Marzo

Abril

► (4) ['Mayo', 'Junio', 'Julio', 'Agosto']

creando arrays a partir de objetos

Se pueden crear arrays a partir de objetos, para ello resulta muy útil, el método *values()* de *Object Methods*.

```
const piezas = {  
  0: 'ruedas',  
  1: 'motor',  
  2: 'puertas',  
  3: 'asientos',  
};  
  
const coche = Object.values(piezas);  
console.log(coche);
```

```
▼ (4) ['ruedas', 'motor', 'puertas', 'asientos']  
  0: "ruedas"  
  1: "motor"  
  2: "puertas"  
  3: "asientos"  
  length: 4  
  ► [[Prototype]]: Array(0)
```


[length]

```
const meses=["Enero","Febrero","Marzo","Abril"];  
console.log(meses.length);
```

Para acceder a todos los elementos de un array, recorrerlo, se utiliza una propiedad del array llamada *length*.

length indica el número de elementos que contiene un array.

Se suele utilizar en un bucle *for* para ir recorriendo los elementos del array (*for* se estudiará con más detenimiento en el próximo tema).

4

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

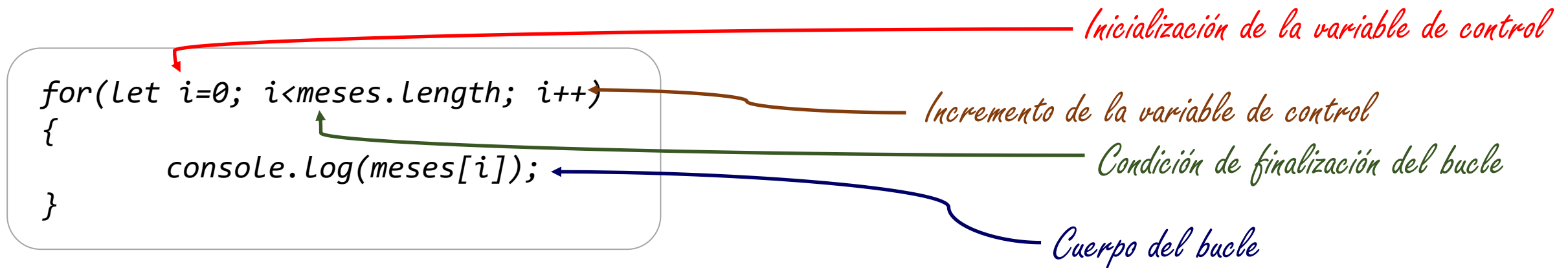
Enero
Febrero
Marzo
Abril

[for]

A continuación de la palabra *for* se ponen paréntesis (), y entre ellos se inicializa la variable que va a controlar las veces que se ejecutan las instrucciones contenidas entre llaves { }, justo seguido de los paréntesis, llamado cuerpo del bucle.

Separada de la inicialización de la variable de control por ; se pone la condición de finalización del bucle. Las instrucciones del cuerpo del bucle se ejecutarán mientras se cumpla dicha condición, que es comprobada al comienzo de cada iteración.

Por último, separado por ; se coloca la instrucción que hace aumentar la variable de control del bucle, se ejecuta al finalizar cada iteración.



[for]

En este caso, como solo hay una instrucción dentro del cuerpo del bucle no hace falta poner las llaves {}.

```
const meses=["Enero","Febrero","Marzo","Abril"];  
for(let i=0; i< meses.length; i++)  
    console.log(meses[i]);
```

Aunque sea menos legible y roce las malas prácticas se puede resumir aún más el bucle.

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

Enero
Febrero
Marzo
Abril

[forEach]

Otro método para iterar o recorrer los arrays es mediante el método *forEach()*.

forEach va a ir recorriendo los elementos del array uno a uno en cada iteración y ejecutando, para cada uno de ellos, una función que se le pasa como argumento.

Esta función recibe como primer parámetro una referencia al elemento del array sobre el que se está trabajando, como segundo argumento el índice de ese elemento en esa iteración y como tercer argumento el propio array.

```
const meses=["Enero","Febrero","Marzo","Abril"];

meses.forEach(function(mes)
{
    console.log(mes)
});
```

Enero
Febrero
Marzo
Abril

[forEach]

Se pueden usar las funciones flecha con *forEach* simplificando muchísimo el código.

```
const meses=["Enero", "Febrero", "Marzo", "Abril"];  
meses.forEach(mes=>console.log(mes));
```

Enero
Febrero
Marzo
Abril

```
meses.forEach((mes, índice)=>console.log(`${mes} en posición ${índice}`));
```

Enero en posición 0
Febrero en posición 1
Marzo en posición 2
Abril en posición 3

[keys]

También se puede utilizar otro método para iterar o recorrer arrays mediante el método *keys()*, que devuelve un iterador, un elemento que contiene la funcionalidad para recorrer el array.

El iterador que devuelve *keys()* va a recorrer las claves de los elementos del array; como los elementos de un array, a diferencia de un objeto literal, no poseen claves, estas hacen referencia a su índice en el array, así el primer elemento tendría como clave el 0, el siguiente el 1, y así sucesivamente.

Los iteradores se estudiarán en profundidad en próximos temas, pero, a modo de introducción a los mismos, el siguiente ejemplo crea un iterador, y cada vez que se ejecuta su método *next()*, pasa a la clave del siguiente elemento y devuelve un objeto literal con dos propiedades, *value* que contiene el valor de dicha clave (el índice) y un booleano llamado *done* que indica si se ha recorrido el array:

```
const meses=["Enero", "Febrero"];  
const iterador = meses.keys();  
console.log(iterador.next());  
console.log(iterador.next());  
console.log(iterador.next());
```

```
▶ {value: 0, done: false}  
▶ {value: 1, done: false}  
▶ {value: undefined, done: true}
```

[values]

Existe otro método llamado *values()*, que devuelve un iterador para recorrer el array (los iteradores se estudiarán en profundidad en próximos temas).

El iterador que devuelve *values()* va a recorrer el valor de los elementos del array.

El siguiente ejemplo crea un iterador, y cada vez que se ejecuta su método *next()*, pasa al valor del siguiente elemento contenido en el array, además, devuelve un objeto literal con dos propiedades, *value* que contiene el elemento y un booleano llamado *done* que indica si se ha recorrido el array:

```
const meses=["Enero","Febrero"];  
const iterador = meses.values();  
console.log(iterador.next());  
console.log(iterador.next());  
console.log(iterador.next());
```

```
▶ {value: 'Enero', done: false}  
▶ {value: 'Febrero', done: false}  
▶ {value: undefined, done: true}
```

[entries]

El método *entries()* devuelve un iterador que contiene pares de índice-valor por cada elemento del array.

Este iterador permite recorrer el valor de los elementos del array.

En el siguiente ejemplo, cada vez que se ejecuta el método *next()* del iterador, creado por *entries()*, se crea un iterador y se pasa al siguiente elemento contenido en el array, además, devuelve un objeto literal con dos propiedades, *value* que contiene un array formado por el índice y el valor del elemento y un booleano llamado *done* que indica si se ha recorrido el array:

```
const meses=["Enero","Febrero"];  
const iterador = meses.entries();  
console.log(iterador.next());
```

```
▼ {value: Array(2), done: false}  
  done: false  
  ▼ value: Array(2)  
    0: 0  
    1: "Enero"  
    length: 2  
    ► [[Prototype]]: Array(0)  
    ► [[Prototype]]: Object
```


[map]

`map()` es otro método que se comporta igual que `forEach`.

```
const meses=["Enero","Febrero","Marzo","Abril"];
meses.map((mes, índice)=>console.log(`${mes} en posición ${índice}`));
```

Enero en posición 0
Febrero en posición 1
Marzo en posición 2
Abril en posición 3

La diferencia de `map` con `forEach` es que `map()` devuelve un array.

```
const resultado=meses.map((mes, índice)=>console.log(`${mes} en posición ${índice}`));
console.log(resultado);
```

```
const otro=meses.forEach((mes, índice)=>console.log(`${mes} en posición ${índice}`));
console.log(otro);
```

En el ejemplo superior se obtendrá *undefined* en salida de la sentencia `console.log(otro);`

[includes]

includes() es un método que se utiliza para comprobar si dentro de un array hay un determinado elemento, si lo incluye.

El método devuelve un booleano, *true* si el elemento se encuentra en el array o *false* si no lo incluye.

```
const meses=["Enero", "Febrero", "Marzo", "Abril"];  
console.log(meses.includes("Febrero"));
```

true

```
const meses=["Enero", "Febrero", "Marzo", "Abril"];  
console.log(meses.includes("febrero"));
```

false

[some]

some() funciona de forma parecida a *includes()* pero es muy útil sobre arrays de objetos.

Este método se utiliza para comprobar si dentro de un objeto que pertenece al array hay un determinado elemento.

En el método hay que definir una función que devolverá un booleano, *true* si existe un determinado elemento que cumpla la condición y *false* si no lo hay. La función recibe como primer argumento el objeto sobre el que está iterando, como segundo argumento el índice de ese objeto en el array, y como tercer elemento el propio array.

```
const meses=[{nombre:"Enero"},{nombre:"Febrero"},  
{nombre:"Marzo"}];  
  
console.log(meses.some(mes=>mes.nombre=="Febrero"));
```

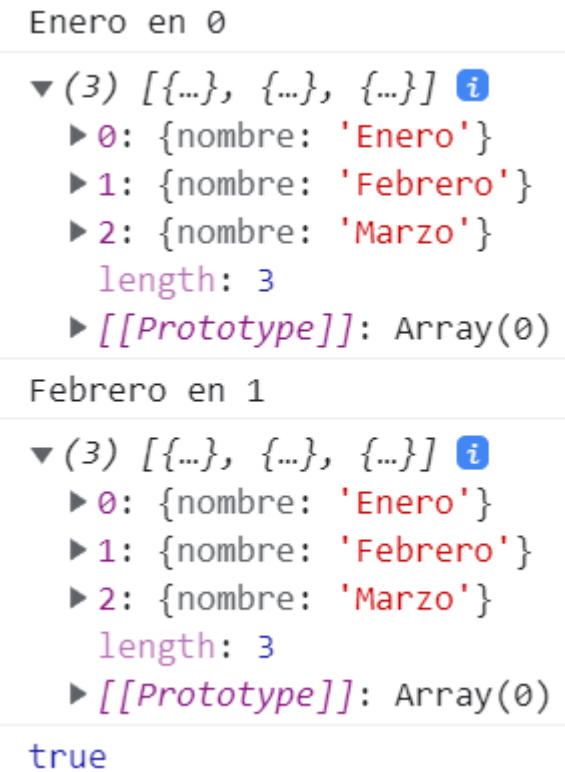
true

[some]

`some()`, al igual que `includes()`, finalizará si encuentra y no seguirá recorriendo el array si encuentra el elemento.

```
const meses=[{nombre:"Enero"},{nombre:"Febrero"},{nombre:"Marzo"}];

console.log(meses.some((mes, indice, miArray)=>
{
    console.log (`${mes.nombre} en ${indice}`);
    console.log(miArray);
    return mes.nombre=="Febrero";}
));
```



```
Enero en 0
▼ (3) [{...}, {...}, {...}] ⓘ
  ► 0: {nombre: 'Enero'}
  ► 1: {nombre: 'Febrero'}
  ► 2: {nombre: 'Marzo'}
  length: 3
  ► [[Prototype]]: Array(0)
Febrero en 1
▼ (3) [{...}, {...}, {...}] ⓘ
  ► 0: {nombre: 'Enero'}
  ► 1: {nombre: 'Febrero'}
  ► 2: {nombre: 'Marzo'}
  length: 3
  ► [[Prototype]]: Array(0)
true
```

[some]

Por último, a modo de ejemplo, se muestra *some* con arrays no de objetos.

```
const meses=["Enero","Febrero","Marzo"];  
console.log(meses.some(mes=>mes=="Febrero"));
```

true

[findIndex]

findIndex() es un método que devuelve el índice en el que se encuentra un determinado elemento de un array.

En el método hay que definir una función que recibe como primer argumento el elemento sobre el que está iterando, como segundo argumento el índice de ese objeto en el array, y como tercer elemento el propio array.

En la función se define la comprobación y devolverá el índice del elemento dentro del array si lo encuentra, o el valor **-1** si no lo encuentra.

```
const meses=["Enero","Febrero","Marzo"];  
console.log(meses.findIndex(mes=>mes=="Febrero"));
```

1

[findIndex]

findIndex() también funciona con arrays de objetos.

```
const meses=[{nombre:"Enero"},{nombre:"Febrero"},  
{nombre:"Marzo"}];  
  
console.log(meses.findIndex((mes, indice, miArray)=>  
{  
    console.log (`${mes.nombre} en ${indice}`);  
    return mes.nombre=="Febrero";}  
));
```

Enero en 0
Febrero en 1
1

[find]

find() devuelve un elemento de un array que cumple una determinada condición.

En el método hay que definir una función que recibe como primer argumento el elemento sobre el que está iterando, como segundo argumento el índice de ese objeto en el array, y como tercer elemento el propio array.

La función retorna en el momento en el que encuentre el primer elemento que cumpla la condición, si llega al final sin encontrar el elemento devolverá *undefined*.

```
const meses=[{nombre:"Enero",días:31},
{nombre:"Febrero",días:28},
{nombre:"Marzo",días:31}];

let mesLargo=meses.find(mes=>mes.días>30);
console.log(mesLargo);
```

```
▼ {nombre: 'Enero', días: 31}
  días: 31
  nombre: "Enero"
  ► [[Prototype]]: Object
```


[filter]

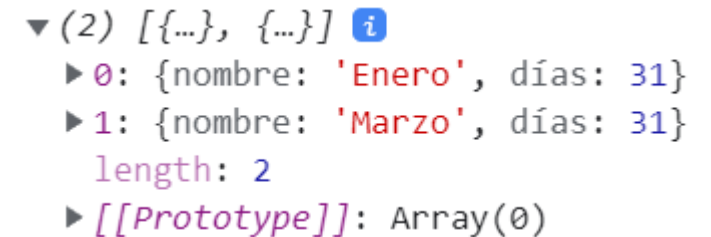
filter() es un método que devuelve los elementos de un array que cumplen una determinada condición, filtra dichos elementos.

En el método hay que definir una función que recibe como primer argumento el elemento sobre el que está iterando, como segundo argumento el índice de ese objeto en el array, y como tercer elemento el propio array.

La función va devolviendo un array con los elementos que cumplen la condición que se defina dentro de la misma. Si no hay elementos cumplan la condición devolverá un array vacío.

```
const meses=[{nombre:"Enero",días:31},
{nombre:"Febrero",días:28},
{nombre:"Marzo",días:31}];

let mesesLargos=meses.filter(mes=>mes.días>30);
console.log(mesesLargos);
```



```
▼ (2) [{...}, {...}] ⓘ
  ► 0: {nombre: 'Enero', días: 31}
  ► 1: {nombre: 'Marzo', días: 31}
    length: 2
  ► [[Prototype]]: Array(0)
```

[every]

every() es un método que devuelve un booleano, *true* si todos los elementos del array cumplen una determinada condición y *false* en caso contrario. Es el método opuesto al método *some()*.

En el método hay que definir una función que recibe como primer argumento el elemento sobre el que está iterando, como segundo argumento el índice de ese objeto en el array, y como tercer elemento el propio array.

```
const meses=[{nombre:"Enero",días:31},  
{nombre:"Febrero",días:28},  
{nombre:"Marzo",días:31}];  
  
console.log(meses.every(mes=>mes.días<=31));
```

true

[reduce]

reduce() es un método reductor, coge una cantidad de datos y devuelve un resumen de los mismos.

En *reduce()* se define, como primer argumento, una función que recibe, como primer argumento, una variable sobre la que se va a trabajar y, como segundo argumento, un nombre de variable que contendrá el elemento del array sobre el que está iterando.

reduce() recibe también, como segundo argumento, el valor inicial de la variable sobre la que se trabaja.

La función devolverá el resultado (el resumen de los datos) del incremento de la instrucción que hay en el cuerpo de la función.

```
const meses=[{nombre:"Enero",días:31},  
{nombre:"Febrero",días:28},  
{nombre:"Marzo",días:31}];  
  
let total=meses.reduce((sumaDias, mes)=>sumaDias+mes.días,0);  
console.log(total);
```

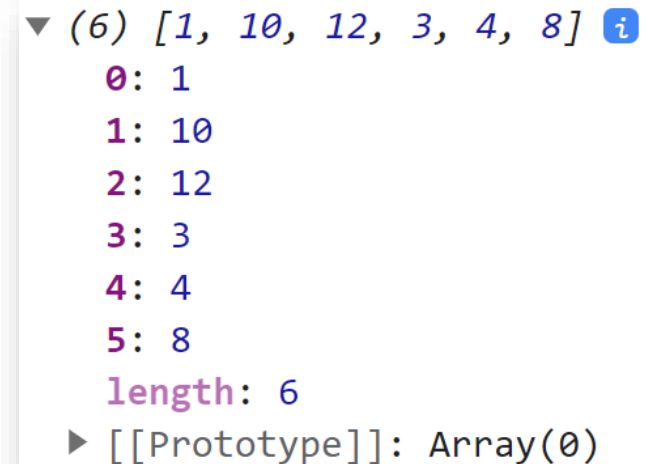
90

[sort y toSorted]

sort() es un método que devuelve un array ordenado, modificando el array original, si no se desea alterar el array original se debe utilizar el método *toSorted()*.

Por defecto, *sort()* y *toSorted()* ordenan alfabéticamente como si el contenido del array fuera texto.

```
const array=[10,12,3,4,8,1];  
console.log(array.sort());
```

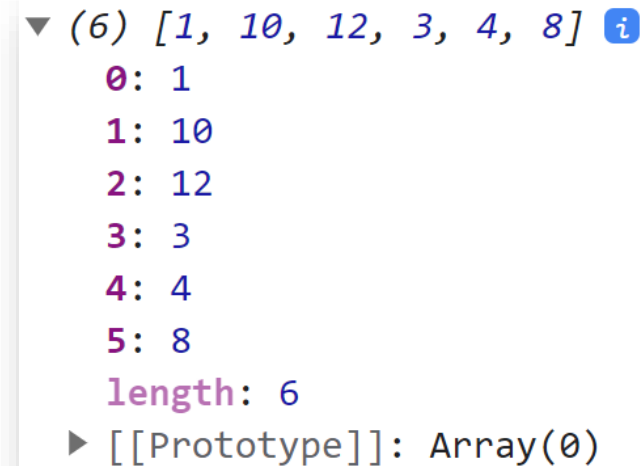


```
▼ (6) [1, 10, 12, 3, 4, 8] ⓘ  
  0: 1  
  1: 10  
  2: 12  
  3: 3  
  4: 4  
  5: 8  
  length: 6  
  ► [[Prototype]]: Array(0)
```

[sort y toSorted]

En el siguiente ejemplo, se realiza una ordenación alfabética, pero en sentido descendiente:

```
const array=[10,12,3,4,8,1];  
console.log(array.sort().reverse());
```



▼ (6) [1, 10, 12, 3, 4, 8] ⓘ

0	1
1	10
2	12
3	3
4	4
5	8

length: 6

► [[Prototype]]: Array(0)

En ejemplo se está utilizando el *Method chaining* (o encadenamiento de métodos, que se estudiará en el próximo tema) para invocar múltiples llamadas a métodos, primero el método `sort()` y, con su resultado, el método `reverse()`.

[sort y toSorted]

`sort()` y `toSorted()` admiten, como argumento, una función comparadora (*compareFn*) que permite indicar como se desea ordenar el array. El valor que devuelva la función comparadora le indicará a `sort()` y `toSorted()` como debe ordenar los elementos del array:

Se mira el signo del valor devuelto por la función comparadora: <i>compareFn(a, b)</i>	Orden de clasificación
<i>Positivo</i> : > 0	Primero va <i>a</i> y después <i>b</i> : $[a, b]$
<i>Negativo</i> : < 0	Primero va <i>b</i> y después <i>a</i> : $[b, a]$
<i>Cero</i> : $=== 0$	Mantiene el orden original de <i>a</i> y <i>b</i>

En el siguiente ejemplo, se utiliza una función de flecha para ordenar de manera numérica en sentido descendente:

```
const array=[10,12,3,4,8,1];  
console.log(array.sort((a,b)=>b-a));
```

```
► (6) [12, 10, 8, 4, 3, 1]
```

Bibliografía y recursos online

- https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Grammar_and_types
- https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array