Arrays

-Crear arrays-

let array = [valor0, valor1, …valorN]; => Crea un array de forma implícita.

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

let array = new Array(n) => Crea un array con N espacios disponibles con undefined como valor.

```
let arr = new Array(5);
console.log(arr);
```

```
[ <5 empty items> ]
```

-Averiguar la longitud de un array-

.length => Devuelve el número de índices que contiene el array.

```
let arr = [1,2,3,4,5];
console.log(arr.length);
```

```
5
```

PROTOTIPOS DE LOS ARRAYS

-Crear arrays con prototipos-

Array.of(valoresAagregar) => Crea un array, con los elementos que le pasemos como argumento.

```
let arr = Array.of(1,2,3,4,5);
console.log(arr);
```

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

-Copiar arrays enteros o secciones de los mismos-

.slice(indiceInicial, indiceFinal) => Devuelve una copia de una porción de un array y la guarda en uno nuevo, por defecto, indiceInicial tendrá un valor de 0, el primer elemento del array, e índice final tendrá el mismo valor que él .length del array que estemos copiando, si indicamos valores negativos en indiceInicial, lo que hará, será tomar como punto de partida el ultimo índice del array, y si indicamos valores negativos en indiceFinal, lo que hará será copiar hasta él .length – el número que indiquemos. indicesAborrar, si este valor es 0 o negativo, no se eliminará ningún elemento del array, simplemente se agregará el valor en el índice especificado, y se empujara al resto de los valores, en caso de no definir el valor, o que este sea más grande que el arr.length-1, se eliminarán todos los elementos que vienen después del índice que especificamos como indiceInicial.

```
let arr = [1,2,3,4,5];
arr.slice(3,5);
```

```
[ 4, 5 ]
```

-Agregar valores-

.push(valor) => Añade lo que pongamos dentro de los paréntesis en el último índice vacío del array

```
let arr = [1,2,3,4,5];
arr.push(10);
console.log(arr);
```

```
6
[ 1, 2, 3, 4, 5, 10 ]
```

.unshift() => agrega lo que pasemos como argumento al primer índice del array

```
let arr = [1,2,3,4,5];
arr.unshift(10);
console.log(arr);
```

```
6
[ 10, 1, 2, 3, 4, 5 ]
```

.fill(valorAutilizar, indiceInicial, indiceFinal) => Rellena un array con el valor que introduzcamos desde el índice que le pongamos, hasta el segundo que especifiquemos, en caso de no aclarar los índices, llenara todo el array, y en caso de que no especifiquemos ningún valor, por defecto, llenara todo el array con "undefined"

```
let arr = [1,2,3,4,5];
arr.fill(15,2,5);
```

```
[ 1, 2, 15, 15, 15 ]
```

-Extraer valores-

.pop() => Extrae y devuelve el ultimo índice ocupado del array, esto puede guardarse en una nueva variable o puede usarse en funciones, no permite argumentos.

```
let arr = [1,2,3,4,5];
arr.pop();
console.log(arr);
```

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

.shift() => Extrae y devuelve el primer índice ocupado del array, esto puede guardarse en una nueva variable o puede usarse en funciones, no permite argumentos.

```
let arr = [1,2,3,4,5];
arr.shift();
console.log(arr);
```

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

-Agregar y borrar valores-

.splice(indiceInicial, indicesAborrar, valor1, … valorN) => Nos permite agregar valores dentro de un array, tenemos que indicarle el índice inicial, que es donde queremos agregar el valor en cuestión, la cantidad de índices a eliminar, en caso de que queramos sobrescribir el valor que se encuentre actualmente en ese índice, y por ultimo los valores que queramos agregar. indiceInicial, puede tomar tanto valores positivos como negativos, esto nos sirve para recorrer el array en una dirección u otra, el valor predeterminado del mismo será 0, el primer índice del array.

```
let arr = [1,2,3,4,5];
arr.splice(2,0,45);
console.log(arr);

arr.splice(4,1,16);
console.log(arr);
```

```
[]
[ 1, 2, 45, 3, 4, 5 ]

[ 4 ]
[ 1, 2, 45, 3, 16, 5 ]
```

-Ordenar valores-

.sort(criterioDeOrdenamiento(primerElemento, segundoElemento)) => Ordena en orden ascendente los elementos dentro del array, si no especificamos una función comparativa como argumento, convertirá los valores del array en strings y los comparará por su valor Unicode, entonces, para hacer que funcione de manera correcta con números, debemos de especificar el criterio que utilizará de la siguiente manera:

```
let arr = [5,4,2,25,0,1,50,3];
arr.sort((a,b) => a-b);
console.log(arr);
```

```
[ 0, 1, 2, 3, 4, 5, 25, 50 ]
[ 0, 1, 2, 3, 4, 5, 25, 50 ]
```

.reverse() => Invierte el orden de los elementos del array, no acepta argumentos.

```
let arr = [1,2,3,4,5];
arr.reverse();
console.log(arr);
```

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

-Chequeando condiciones-

Array.isArray(arrayAchequear) => Comprueba si lo pasado como argumento es un array, devuelve true o false

```
let arr1 = [1,2,3,4,5];
let arr2 = "hola";
let arr3 = {nombre: "Mauri", raza: "Dobberman"};

Array.isArray(arr1);
Array.isArray(arr2);
Array.isArray(arr3);
```

```
true
false
false
```

.includes(valorAchequear) => Chequea si el argumento que pasamos se encuentra dentro del array, devuelve true o false, solo chequea por nivel, o sea, que si dentro del array, tenemos arrays, tendremos que especificar la posición donde se encuentra el array anidado para chequear dentro del mismo

```
let arr = [1,2,"hola",3,4,5];

arr.includes(5);                    true
arr.includes(255);                  false
arr.includes("hola");               true
```

.every(funcionComparativa(elemento, indice, array)) => Comprueba la función que le pasamos a cada uno de los argumentos del array, si todos pasan la comprobación devuelve true, de lo contrario devuelve false, no da bien cuando se trabaja con arrays anidados, si el array anidado tiene más de un índice, cuando llamamos a la función comparativa, no debemos usar los paréntesis para llamarla.

```
let arr = [2,1,3,2,5];
let arr2 = ["a",2,"hola",5];
let arr3 = ["hola","array","prueba"];

let compStr = (arg) => (typeof arg === "string");
let compNum = (arg) => (typeof arg === "number");

arr.every(compNum);                 true
arr2.every(compNum);                false
arr3.every(compNum);                false

arr.every(compStr);                 false
arr2.every(compStr);                false
arr3.every(compStr);                true
```

Podemos también, especificar la condición dentro del paréntesis de la siguiente manera

```
let arr = [2,1,3,2,5];
let arr2 = ["a",2,"hola",5];
let arr3 = ["hola","array","prueba"];

arr.every((arg) => (typeof arg === "number"));      true
arr2.every((arg) => (typeof arg === "number"));     false
arr3.every((arg) => (typeof arg === "number"));     false

arr.every((arg) => (typeof arg === "string"));      false
arr2.every((arg) => (typeof arg === "string"));     false
arr3.every((arg) => (typeof arg === "string"));     true
```

.some(funcionComparativa(elemento, indice, array)) => Funciona como el every, pero en vez de chequear que se cumpla la condición en todos los elementos del array, basta con que un elemento la cumpla para devolver true, en caso de que ninguno la cumpla, devolverá false.

```
let arr = [2,1,3,2,5];
let arr2 = ["a",2,"hola",5];
let arr3 = ["hola","array","prueba"];

let compNum = (arg) => (typeof arg === "number");
let compStr = (arg) => (typeof arg === "string");

arr.some(compNum);                                  true
arr2.some(compNum);                                 true
arr3.some(compNum);                                 false

arr.some(compNum);                                  true
arr2.some(compNum);                                 true
arr3.some(compNum);                                 false
```

Como en el every, también podemos definir la condición dentro del paréntesis:

```
let arr = [2,1,3,2,5];
let arr2 = ["a",2,"hola",5];
let arr3 = ["hola","array","prueba"];

arr.some((arg) => (typeof arg === "number"));      true
arr2.some((arg) => (typeof arg === "number"));     true
arr3.some((arg) => (typeof arg === "number"));     false

arr.some((arg) => (typeof arg === "string"));      false
arr2.some((arg) => (typeof arg === "string"));     true
arr3.some((arg) => (typeof arg === "string"));     true
```

-Conversión de valores-

.toString() => Convierte todos los índices del array en una string, no permite argumentos

```
let arr = [1,2,"hola",3,4,5];
arr.toString();                  '1,2,hola,3,4,5'
console.log(arr);                [ 1, 2, 'hola', 3, 4, 5 ]
```

.toLocaleString(localidad, {opciones}) => Devuelve una cadena de texto representando los elementos del array, la conversión a texto depende del valor "local" que hayamos definido, tanto en formato, representación y separación, por ejemplo, la fecha 04/05/2022 se escribe como 05/04/2022 en EEUU, funciona también con monedas.

```
let fecha = new Date("03 mayo 2022 22:59 UTC-3");
console.log(fecha);                                  2022-05-04T01:59:00.000Z

let array = [fecha];
console.log(array);                                  [ 2022-05-04T01:59:00.000Z ]

let result = array.toLocaleString("es", {timeZone: "UTC"});
console.log(result)                                  '4/5/2022 1:59:00'

let result2 = array.toLocaleString("en", {timeZone: "UTC"})
console.log(result2);                                '5/4/2022, 1:59:00 AM'
```

-Iteradores-

.forEach(callback(valorActual, indiceActual, array)) => aplica una función que pasemos como argumento(a las cuales les va a aplicar 3 argumentos, el valor actual, el índice actual y el array en cuestión) a cada elemento del array, realiza lo que la función callback le haya pedido y no retorna nada como resultado, o sea que, no podremos guardar su resultado en una variable, sino que con esta podremos asignar valores a variables ya existentes fuera del forEach

```
let arr = [2,1,3,2,5];

arr.forEach((a,b) => console.log("El número " + a +
" se encuentra en el índice " + b));
```

```
'El número 2 se encuentra en el índice 0'
'El número 1 se encuentra en el índice 1'
'El número 3 se encuentra en el índice 2'
'El número 2 se encuentra en el índice 3'
'El número 5 se encuentra en el índice 4'
```

```
let pares = {};
arr.forEach((a,b) => pares[b] = a);
console.log(pares);
```

```
{ '0': 2, '1': 1, '2': 3, '3': 2, '4': 5 }
```

.map(callback(valorActual, indiceActual, array)) => Funciona de la misma manera que el forEach, pero a diferencia de la anterior, esta si retorna un nuevo array con los resultados de la función callback que hayamos ingresado como parámetro.

```
let arr = [2,1,3,2,5];

//Diferencia con forEach
console.log(arr.forEach((a,b) => a* b));
arr.forEach((a,b) => a*b);
console.log(arr);
//aplicando .map
arr = arr.map((a,b) => a*b);
console.log(arr);
```

```
undefined
[ 2, 1, 3, 2, 5 ]
[ 0, 1, 6, 6, 20 ]
[ 0, 1, 6, 6, 20 ]
```

.reduce(callback(acumulador, valorActual, indiceActual, array)) => Itera sobre todo el array y aplica una función callback que hayamos pasado como argumento, devolviendo un único valor como resultante, la función recibe cuatro argumentos, el acumulador, que es donde iremos guardando el valor resultante de las iteraciones, el valor actual, que es el valor en el que se encuentra iterando, el índice actual, y el array en cuestión

```
let arr = [2,1,3,2,5];
let arr2 = ["hola ", "soy ", " Alan"]

arr.reduce((a, b) => a+b);
arr2.reduce((a,b) => a+b);
```

```
13
'hola soy  Alan'
```

.reduceRight(callback(acumulador, valorActual, indiceActual, array)) => Lo mismo que el reduce, pero en sentido inverso.

```
let arr = [" Hola "," soy "," alan "];

arr.reduceRight((a,b)=> a+b);
```

```
' alan  soy  Hola '
```

.filter(callback(valorActual, indiceActual, array)) => Llama a la función callback sobre cada elemento del array y construye un nuevo array con todos los elementos para los cual el resultado de callback es true, la función solo es invocada para los índices que tengan un valor asignado, en caso de que no lo tengan, el callback los salteará y no serán incluidos en el nuevo array

```
let arr = [2,1,"hola",3,"string",2,5];

arr.filter((a) => typeof a === "string");
console.log(arr);

let arr2 = arr.filter((a) => typeof a ==="number")
console.log(arr2)
```

```
[ 'hola', 'string' ]
[ 2, 1, 'hola', 3, 'string', 2, 5 ]

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

-Búsqueda-

.indexOf(valorAbuscar, indiceInicial) => Busca el primer argumento que le pasemos, a partir del índice que indiquemos, si el número es positivo, será la posición desde la que empezara a buscar, si el índice es negativo, tomará las posiciones que le indiquemos restándole el número al .length, o sea, desde atrás hacia adelante, en caso de no indicar índice, lo hará desde el índice 0, o sea, buscará en el array completo, en caso de encontrar el valor, nos devuelve el índice en el que se encuentra, en caso de no encontrarlo, devuelve -1, solo chequea por nivel, o sea, que si dentro del array, tenemos arrays, tendremos que especificar la posición donde se encuentra el array anidado para chequear dentro del mismo

```
let arr = [2,1,"hola",3,"string",2,5];

arr.indexOf("hola");          2
arr.indexOf(3);               3
arr.indexOf(5);               6
arr.indexOf(2);               0
arr.indexOf(2, -1);           -1
```

.find(funcionComparativa(elemento, indice, array)) => Devuelve el primer valor que encuentre en el array que cumpla con la condición de la función comparadora que pongamos como parámetro.

```
let arr = [2,1,25,3,56,2,5];

arr.find(a => a > 45);        56
```

.findIndex(funcionComparativa(elemento, indice, array)) => Devuelve el indice del primer elemento del array que cumpla con la función comparadora proporcionada.

```
let arr = [2,1,"Hola",25,"JS",3,17,22];

arr.findIndex(a => typeof a === "string");    2
```

.lastIndexOf(elementoAbuscar) => Devuelve el ultimo índice en el que se encuentra el valor que ingresamos como argumento, en caso de que el array tenga varios elementos con el mismo valor, esta función nos devolverá el ultimo índice en el que se encuentre dicho valor.

```
let arr = [2,1,25,3,56,2,5];

arr.lastIndexOf(2);                    5
```

-Concatenadores-

arr0.concat(arr1, arr2 … arrN) => Une dos o más arrays, este método no cambia los arrays existentes, sino que crea uno enteramente nuevo, con los valores que le demos como argumento

```
let arr = [2,1,25,3];
let arr2= [2,15,10];
let arr3= [20,10,30];
arr2.concat(arr, arr3);          [ 2, 15, 10, 2, 1, 25, 3, 20, 10, 30 ]
```

.join() => Crea y devuelve una nueva string concatenando todos los valores de un array, separados por lo que le especifiquemos en su argumento.

```
let arr = [2,1,"Hola",25,"JavaScript",3];
arr.join(" ");                    '2 1 Hola 25 JavaScript 3'
arr.join("_");                    '2_1_Hola_25_JavaScript_3'
arr.join(" ; ");                  '2 ; 1 ; Hola ; 25 ; JavaScript ; 3'
```

.copyWithin(indice, indiceInicial, indiceFinal) => Transfiere una copia plana de una sección a otra dentro del mismo array sin modificar su propiedad length y lo devuelve. Recibe tres argumentos, índice, que es índice inicial donde “pegaremos” nuestra sección definida (este puede ser tanto positivo como negativo, en el caso de definirlo con un numero negativo, lo que hará es comenzar desde el final hacia el inicio), indiceInicial, que es el índice a partir del cual comenzará la copia de los elementos (si no es definido, se tomara 0 como valor por defecto) y el índice final, que es donde acabara la copia (si no es definido, tomara el length del array como valor).

```
let arr = [2,1,"Hola",25,"JS",3,17,22];

arr.copyWithin(5, 2, 5);          [ 2, 1, 'Hola', 25, 'JS', 'Hola', 25, 'JS' ]
```

-Objetos iterables-

.values() => Devuelve un nuevo objeto iterable, que contiene los valores de cada índice del array, algo a tener en cuenta sobre este prototipo es que el resultado es como una lista enlazada, pero tratando al nodo next como una función.

```
let arr = ["Mauri",17,"Fideos",3,false,18];

let iterador = arr.values();

console.log(iterador.next());                { value: 'Mauri', done: false }
console.log(iterador.next());                { value: 17, done: false }
console.log(iterador.next());                { value: 'Fideos', done: false }
console.log(iterador.next().value);          3

let prueba = iterador.next();
let prueba2 = iterador.next().value;
console.log(prueba);                         { value: false, done: false }
console.log(prueba2);                        18
```

.entries() => Devuelve un nuevo objeto iterable, que contiene los pares indice/valor que conforman al array en su totalidad, es prácticamente igual al values, pero este no solo nos devuelve el valor, sino que, por cada valor, nos devuelve un array con su par indice/valor.

```
let arr = ["Mauri",17,"Fideos",3,false,18];

let iterador = arr.entries();

console.log(iterador.next());                { value: [ 0, 'Mauri' ], done: false }
console.log(iterador.next());                { value: [ 1, 17 ], done: false }
console.log(iterador.next());                { value: [ 2, 'Fideos' ], done: false }
console.log(iterador.next().value);          [ 3, 3 ]

let prueba = iterador.next();
let prueba2 = iterador.next().value;
console.log(prueba);                         { value: [ 4, false ], done: false }
console.log(prueba2);                        [ 5, 18 ]
```

.keys() => Devuelve un nuevo objeto iterable que contiene los índices con las que se puede acceder a cada elemento del array

```
let arr = ["Mauri",17,"Fideos",3,false,18];

let iterador = arr.keys();

console.log(iterador.next());                { value: 0, done: false }
console.log(iterador.next());                { value: 1, done: false }
console.log(iterador.next());                { value: 2, done: false }
console.log(iterador.next().value);          3

let prueba = iterador.next();
let prueba2 = iterador.next().value;
console.log(prueba);                         { value: 4, done: false }
console.log(prueba2);                        5
```

.from() => Este nos permite convertir un objeto iterable en un array

```
let arr = ["Mauri",17,"Fideos",3,false,18];

let iterador = arr.entries();
console.log(iterador);                       Object [Array Iterator] {
                                               __proto__: { next: ƒ next() }
                                             }

let arrayNuevo = Array.from(iterador);
console.log(arrayNuevo);                     [
                                               [ 0, 'Mauri' ],
                                               [ 1, 17 ],
                                               [ 2, 'Fideos' ],
                                               [ 3, 3 ],
                                               [ 4, false ],
                                               [ 5, 18 ]
                                             ]
```

-Reducir profundidad-

.flat(profundidad) => Devuelve un array con todos sus elementos y los de sus sub-arrays en el mismo nivel, para esto, debemos indicarle la profundidad que queramos como parámetro, la cual tiene un valor predeterminado de 1.

```
let arr = [17,3,[15,["hola","mundo"]],false,18];
console.log(arr);
let arrayFlat = arr.flat(2);

console.log(arrayFlat);
```

```
[ 17, 3, [ 15, [ 'hola', 'mundo' ] ], false, 18 ]

[ 17, 3, 15, 'hola', 'mundo', false, 18 ]
```

.flatMap(callback(valorActual, indiceActual, array)) => Mapea cada elemento y luego aplana el resultado en un nuevo array, esto es lo mismo que aplicar un .map() y luego un .flat(1)

```
//Con map y flat
let arr = [1,2,3,4,5];
arr = arr.map(a => [ a * 5 ]);
arr = arr.flat();
console.log(arr);

//con flatMap
let arr2 = [1,2,3,4,5];
arr2 = arr2.flatMap(a => [ a * 5 ])
console.log(arr2);
```

```
[ [ 5 ], [ 10 ], [ 15 ], [ 20 ], [ 25 ] ]
[ 5, 10, 15, 20, 25 ]
[ 5, 10, 15, 20, 25 ]

[ 5, 10, 15, 20, 25 ]
[ 5, 10, 15, 20, 25 ]
```