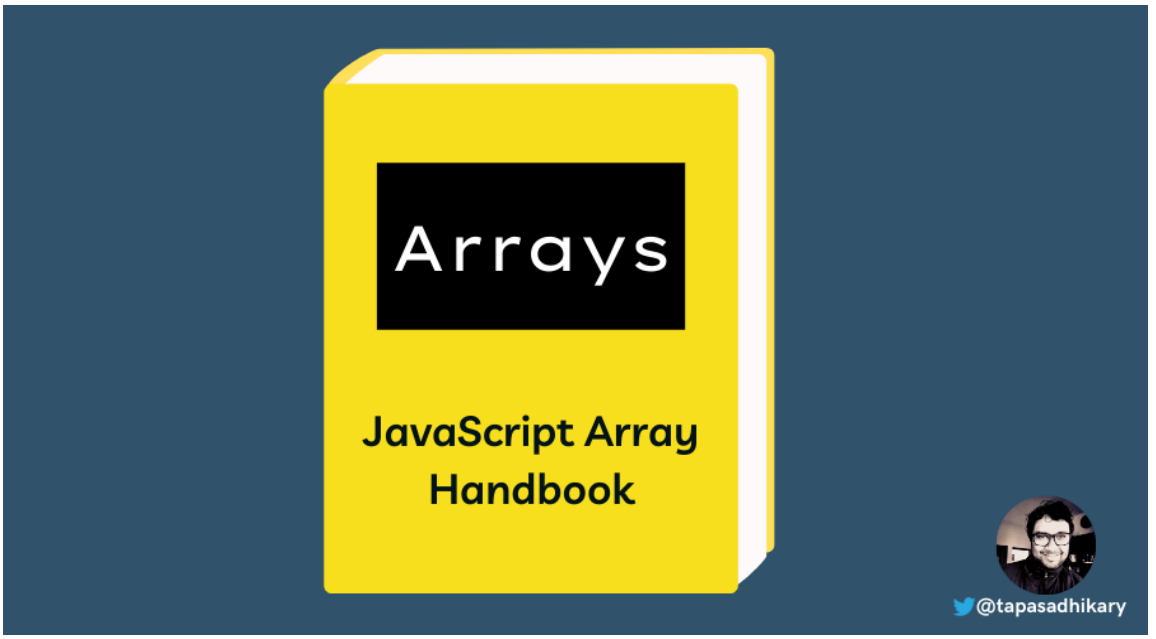
Manual de arrays en JavaScript: Métodos de arrays de JS explicados con ejemplos

Fuente de la captura: Manual de arrays en JavaScript: Métodos de arrays de JS explicados con ejemplos

Manual de arrays en JavaScript: Métodos de arrays de JS explicados con ejemplos

Sil Zubikarai



Artículo original escrito por:TAPAS ADHIKARY**Artículo original:**The JavaScript Array Handbook – JS Array Methods Explained with Examples**Traducido y adaptado por:****Sil Zubikarai**

En programación, un `array` es una colección de elementos o cosas. Los arrays guardan data como elementos y los regresan cuando los necesitas.

La estructura de datos de array es ampliamente usada en todos los lenguajes de programación que la soportan.

En este manual, te enseñará todo acerca de los arrays en JavaScript. Aprenderás acerca de lo complejo del manejo de datos, destructurando, los métodos de array más comúnmente usados, y más.

¿Qué es un array en JavaScript?

Un par de `square brackets []` representa un array en JavaScript. Todos los elementos en un array están separados por una `comma(,)`.

En JavaScript, los arrays pueden ser una colección de elementos de cualquier tipo. Esto significa que tú puedes crear un array con elementos de tipo Cadena , Boolean, Número, Objetos, e incluso otros arrays.

Aquí hay un ejemplo de un array con otros cuatro elementos: tipo Número, Boolean, Cadena y Objecto.

```
const mixedTypedArray = [100, true, 'freeCodeCamp', {}];
```

La posición de un elemento en el array es conocido como `indice`. En JavaScript, el indice del array empieza con `0`, e incrementa uno a uno con cada elemento.

Entonces, por ejemplo, en el array de arriba, el elemento 100 es en `indice 0`, cierto si está en `indice 1`, 'freeCodeCamp' está en `indice 2`, y así.

El número de elementos en el array determina su longitud. Por ejemplo, la longitud del array de arriba es cuatro.

Curiosamente, los arrays de JavaScript no tienen longitud fija. Tú puedes cambiar la longitud en cualquier momento asignando un valor numérico positivo. Aprenderemos más acerca de esto dentro de poco.

Como crear un array en JavaScript

Tú puedes crear un array de diferentes formas en JavaScript. La forma más sencilla es asignar un valor de array a una variable.

```
const salad = ['', '', '', '', '', '', ''];
```

También puedes usar el constructor de array para crear un array.

```
const salad = new Array('', '', '', '', '', '', '');
```

> *Tenga en cuenta:* `new Array(2)` *creará un array de longitud 2 y ninguno de los elementos son definidos en él. Sin embargo,* `new Array(1,2)` *creará un array de longitud dos con elementos 1 y 2 en él.*

Hay otros métodos como `Array.of()` y `Array.from()`, y el `spread` operador(`...`) que te ayuda a crear arrays, también. Aprenderemos acerca de ellos después en este artículo.

Como obtener elementos de un array en JS

Tu puedes acceder y traer elementos de un array usando su indice. Tú necesitas usar la sintaxis `square bracket` para acceder a los elementos del array.

```
const element = array[index];
```

Según sus casos de uso, tú debes escoger acceder a los elementos del array uno por uno o en un bucle.

Cuando accedes a elementos usando un indice como este:

```
const salad = ['', '', '', '', '', '', ''];
salad[0]; // ''
salad[2]; // ''
salad[5]; // ''
```

Puedes usar la longitud de un array para retroceder y acceder elementos.

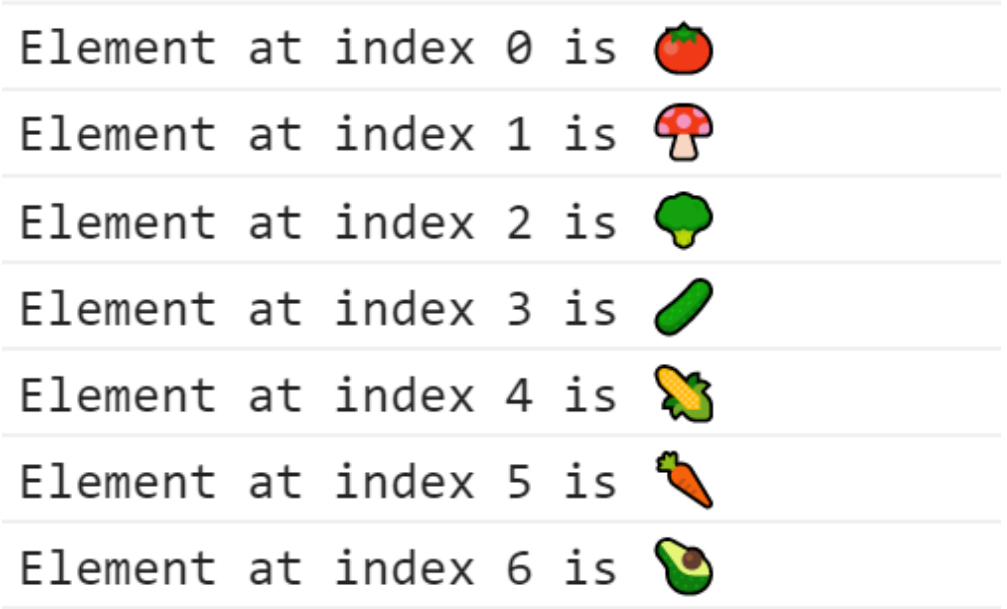
```
const salad = ['', '', '', '', '', '', ''];
const len = salad.length;
salad[len - 1]; // ''
salad[len - 3]; // ''
```

También puedes iterar a través del array usando el común bucle `for` o `forEach`, o cualquier otro bucle.

```
const salad = ['', '', '', '', '', '', ''];

for(let i=0; i<salad.length; i++) {
console.log(`Element at index ${i} is ${salad[i]}`);
}
```

Y aquí esta el resultado:



Como añadir elementos al array en JS

Usa el método `push()` para añadir un elemento en el array. El método `push()` añade un elemento al final del array. Ve como añadimos algunos cacahuetes a la ensalada, como esto:

```
const salad = ['', '', '', '', '', '', ''];
salad.push('');
```

Ahora el array salad es:

["", "", "", "", "", "", "", ""]

Nota que el método `push()` añade un elemento al final del array. Si tu quieres añadir un elemento al inicio del array, vas a necesitar usar el método `unshift()`.

```
const salad = ['', '', '', '', '', '', ''];
salad.unshift('');
```

Ahora el array salad es:

["", "", "", "", "", "", "", ""]

Como eliminar elementos de un array en JS

La manera más sencilla de eliminar un solo elemento de un array usando el método `pop()` . Cada vez que llamas el método `pop()` , este elimina un elemento del final de un array. Entonces este regresa con el elemento eliminado y cambia el array original.

```
const salad = ['', '', '', '', '', '', ''];
salad.pop(); //

console.log(salad); // ['', '', '', '', '', '']
```

Usa el método `shift()` para eliminar un elemento desde el principio del array. Como el método `pop()` , `shift()` regresa el elemento eliminado y cambia el array original.

```
const ensalada = ['', '', '', '', '', '', ''];
salad.shift(); //

console.log(ensalada); // ['', '', '', '', '', ''];
```

Como copiar y clonar un array en JS

Tu puedes copiar y clonar un array a un nuevo array usando el metodo `slice()` . Vea que el método `slice()` no cambia el array original. En cambio crea una nueva copia superficial del array original.

```
const ensalada = ['', '', '', '', '', '', ''];
const ensaladaCopy = ensalada.slice();

console.log(ensaladaCopy); // ['', '', '', '', '', '', '']

ensalada === ensaladaCopy; // returns false
```

Alternativamente, tú puedes usar el operador `spread` para crear una copia del array. Aprenderemos sobre eso pronto.

Como determinar si el valor es un array en JS

Tú puedes determinar si un valor es un array usando el método `Array.isArray(value)`. El método regresa verdadero si el valor que pasa es un array.

```
Array.isArray(['', '', '', '', '', '', '']); // returns true
Array.isArray(''); // returns false
Array.isArray({ 'tomate': ''}); // returns false
Array.isArray([]); // returns true
```

Desestructuración de arrays en JavaScript

Con ECMAScript 6 (ES6), tenemos una nueva sintaxis para extraer múltiples propiedades de un array y asignarlas a variables de una sola vez. Es útil para ayudar a mantener tu código limpio y conciso. Esta nueva sintaxis es llamada sintaxis de desestructuración.

Aquí hay un ejemplo de extraer valores de un array usando la sintaxis de desestructuración:

```
let [tomate, hongo, zanahoria] = ['', '', ''];
```

Ahora tú puedes usar las variables en tu código:

```
console.log(tomate, hongo, zanahoria); // Output,
```

Para hacer lo mismo sin desestructurar, se vería como esto:

```
let vegetales = ['', '', ''];
let tomate = vegetales[0];
let hongo= vegetales[1];
let zanahoria= vegetales[2];
```

Así, la sintaxis desestructurar te salva de escribir un montón de código. Esto le da un gran impulso a la productividad.

Como asignar un valor por defecto a una variable

Tú puedes asignar un valor por defecto usando desestructuración cuando no hay valor o es `undefined` para el elemento array

En el ejemplo de abajo, asignamos por defecto el valor de la variable hongos.

```
let [tomate , hongos = ''] = [''];
console.log(tomate); // ''
console.log(hongos); // ''
```

Como Saltar el Valor en un array

Cuando estás desestructurando, tú puedes saltar un elemento de un array a un variable. Por ejemplo, puedes no estar interesado en todos los elementos de un array. En este caso, saltarse un valor es útil.

En el ejemplo de abajo, nosotros saltamos el elemento hongo. Note el espacio en la declaración de variable en el lado izquierdo de la expresión.

```
let [tomate, , zanahoria] = ['', '', ''];

console.log(tomate); // ''
console.log(zanahoria); // ''
```

Desestructurando un array Anidado en JS

En JavaScript, los arrays se pueden anidar. Esto significa que un array puede tener otro array como elemento. Él anidamiento de array puede ir a más profundidad.

Por ejemplo, vamos a crear un array anidado para frutas. El tiene pocas frutas y un array de vegetales en él.

```
let frutas = ['', '', '', '', ['', '', '']];
```

Como puedes acceder a '' desde el array de arriba. De nuevo, tú puedes hacer esto desestructurando, como esto:

```
const veg = frutas[4]; // returns the array ['', '', '']
const zanahoria = veg[2]; // returns ''
```

Alternativamente, tú puedes usar esta sintaxis corta:

```
fruits[4][2]; // returns ''
```

Tú también puede acceder usando la sintaxis de desestructurado, como esto:

```
let [,,,,[,,zanahoria]] = ['', '', '', '', ['', '', '']];
```

Como usar la sintaxis Spread y el resto de parámetros en JavaScript

Desde ES6, podemos usar el `...` (si, tres puntos consecutivos) como sintaxis spread y el resto de los parámetros en la desestructuración de array.

- Por el resto del parámetro, el `...` aparece en el lado izquierdo de desestructuración.
- Para la sintaxis spread, el `...` aparece en el lado derecho de la desestructuración.

Como usar el resto del parámetro en JS

Con el Resto del Parámetro, podemos organizar los elementos de la izquierda de un array en un nuevo array. El resto de los parámetros deben ser la última variable en la sintaxis de desestructuración.

En el ejemplo de abajo, tenemos organizados los dos primeros elementos de un array a las variables tomate y hongo. El resto de los elementos son organizados a la variable `rest` usando el `...`. La variable `rest` es un nuevo array conteniendo los elementos sobrantes.

```
const [tomate, hongo, ...rest] = ['', '', '', '', '', '', ''];

console.log(tomate); // ''
console.log(hongo); // ''
console.log(rest); // ["", "", "", "", ""]
```

Como usar el Operador Spread en JS

Con el operador spread, podemos crear un clon/copia del array existente como este:

```
const ensalada = ['', '', '', '', '', '', ''];

const ensaladaCloned = [...ensalada];
console.log(ensaladaCloned); // ["", "", "", "", "", "", ""]

ensalada === ensaladaCloned // false
```

Desestructuración de Casos de uso en JavaScript

Vamos a ver algunos emocionantes usos-casos de desestructuración de array, el operador spread, y el resto del parámetro.

Como cambiar los Valores con Desestructuración

Podemos cambiar el valor de dos variables fácilmente usando la sintaxis de desestructuración de array.

```
let primero = '';
let segundo = '';
[primero, segundo] = [segundo, primero];

console.log(primero); // ''
console.log(segundo); // ''
```

Como combinar dos arrays

Podemos combinar dos arrays y crear un nuevo array con todos los elementos de ambos arrays. Vamos a tomar dos arrays– uno con un par de caras sonrientes y otro con algunos vegetales.

```
const emotion = ['', ''];
const veggies = ['', '', '', ''];
```

Ahora, podemos combinarlos para crear un nuevo array.

```
const emotionalVeggies = [...emotion, ...veggies];
console.log(emotionalVeggies); // ["", "", "", "", "", ""]
```

Métodos de array JavaScript

Hasta ahora, hemos visto algunas propiedades de arrays y métodos. Vamos a recapitular rápidamente los que hemos visto.

- `push()` – Inserta un elemento al final del array.

- `unshift()` – Inserta un elemento al inicio del array.
- `pop()` – Remueve un elemento del final del array.
- `shift()` – Remueve un elemento del principio del array.
- `slice()` – Crea una copia sombra del array.
- `Array.isArray()` – Determina si el valor es un array.
- `length` – Determina el tamaño del array.

Ahora aprenderemos sobre otros importantes métodos de array JS con ejemplos.

Como Crear, Remover, Actualizar, y Acceder arrays en JavaScript.

En esta sección, aprenderemos acerca de métodos que puedas usar para crear un nuevo array, remover elementos para hacer un array vacío, acceder elementos, y mucho más.

El método de array concat()

El método `concat()` combina uno o más arrays y regresa un array combinado. Es un método inmutable. Esto significa que no cambia(muta) un array existente.

Vamos a concatenar dos arrays.

```
const first = [1, 2, 3];
const second = [4, 5, 6];

const merged = first.concat(second);

console.log(merged); // [1, 2, 3, 4, 5, 6]
console.log(first); // [1, 2, 3]
console.log(second); // [4, 5, 6]
```

Usando el método `concat()` podemos combinar más de dos arrays. Podemos combinar cualquier número de arrays con esta sintaxis:

```
array.concat(arr1, arr2,..,..,..,arrN);
```

Aquí hay un ejemplo:

```
const first = [1, 2, 3];
const second = [4, 5, 6];
const third = [7, 8, 9];
```

```
const merged = first.concat(second, third);

console.log(merged); // [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

El método de array join()

El método `join()` junta todos los elementos de un array usando un separador y regresa una cadena. El separador por defecto usado para juntar es `comma(,)`.

```
const emotions = ['', '', '', ''];

const joined = emotions.join();
console.log(joined); // ",,,"
```

Tú puedes pasar un separador de tu elección para juntar elementos. Aquí es un ejemplo de juntar elementos con un separador personalizado.

```
const joined = emotions.join('<=>');
console.log(joined); // "<=><=><=>"
```

Invocando el método `join()` en un array vacío regresa una cadena vacía:

```
[].join() // returns ""
```

El método de array fill()

El método `fill()` llena un array con un valor estático. Tú puedes cambiar todos los elementos a valores estáticos o cambiar algunos pocos elementos seleccionados. Nota el método `fill()` cambia el array original.

```
const colors = ['red', 'blue', 'green'];

colors.fill('pink');
console.log(colors); // ["pink", "pink", "pink"]
```

Aquí hay un ejemplo donde cambiamos solo los dos últimos elementos de un array usando el método `fill()`:

```
const colors = ['red', 'blue', 'green'];

colors.fill('pink', 1,3); // ["red", "pink", "pink"]
```

En este caso, el primer argumento del método `fill()` es el valor con el que cambiamos. El segundo argumento es el índice de inicio que se va a cambiar. Este empeiza con `0`. El último argumento es para determinar donde dejar de llenar. El valor maximo podria ser `colors.length`.

Por favor, checa esto en este hilo de Twitter para un uso práctico del método `fill()`.

También, puedes encontrar este demo proyecto de ayuda: https://github.com/atapas/array-fill-color-cards.

El método de array includes()

Tú puedes determinar la presencia de un elemento en un array usando el método `includes()`. Si un elemento es encontrado, el método regresa `true`, y de otra forma `false`.

```
const names = ['tom', 'alex', 'bob', 'john'];

names.includes('tom'); // returns true
names.includes('july'); // returns false
```

El método de array indexOf()

Si tú quieres saber la posición índice de un elemento array. Tú puedes usar el método `indexOf()` para obtener eso. Devuelve el índice de la primera aparición de un elemento en el array. Si un elemento no se encuentra, el método `indexOf()` regresa `-1`.

```
const nombres = ['tom', 'alex', 'bob', 'john'];

nombres.indexOf('alex'); // returns 1
nombres.indexOf('rob'); // returns -1
```

Hay otro método `lastIndexOf()` esto ayuda a encontrar el índice de la última aparición de un elemento en un array. Como `indexOf()`, `lastIndexOf()` también regresa `-1` cuando el elemento no es encontrado.

```
const nombres = ['tom', 'alex', 'bob', 'tom'];

nombres.indexOf('tom'); // returns 0
nombres.lastIndexOf('tom'); // returns 3
```

El método de array reverse()

Como el nombre sugiere, el método `reverse()` reserva la posición del elemento en un array entonces el último elemento va en la primera posición y la primera en la última.

```
const nombres = ['tom', 'alex', 'bob'];

nombres.reverse(); // returns ["bob", "alex", "tom"]
```

El método `reverse()` modifica el array original.

El método de array sort()

El método `sort()` es probablemente uno de los métodos más usados de array. El método por predeterminado `sort()` convierte los tipos de elemento en cadena y luego los arregla. El orden de clasificación predeterminado es ascendente. El método sort () cambia la matriz original.

```
const nombres = ['tom', 'alex', 'bob'];

nombres.sort(); // returns ["alex", "bob", "tom"]
```

El método sort () acepta una función de comparación opcional como argumento. Puedes escribir una función de comparación y pasar al método sort () para anular el comportamiento de clasificación predeterminado.

Vamos a tomar un array de números y ordenarlos de manera ascendente y descendente con la función sort.

```
const numeros = [23, 5, 100, 56, 9, 13, 37, 10, 1]
```

Primero, podemos invocar el método por defecto `sort()` y ver el output:

```
numeros.sort();
```

Ahora el orden del array es , [1,10,100, 13, 23, 37, 5, 56, 9]. Bien eso no es el resultado esperado. Pero esto pasa por que el método default `sort()` convierte los elementos de

la cadena de texto y luego lo compara basado en los valores de la unidad de código `UTF-16`

Para resolver esto, vamos a escribir la funcion comparadora. Aqui hay una para el orden ascendente:

```
function ascendingComp(a, b){
return (a-b);
}
```

Ahora pasa esto por el método `sort()`:

```
numeros.sort(ascendingComp); // retruns [1, 5, 9, 10, 13, 23, 37, 56, 100]

/*

También podemos codificarlo como,

numeros.sort(function(a, b) {
return (a-b);
});

O, con la funcion arrow,

numeros.sort((a, b) => (a-b));

*/
```

Para el orden descendente, haz esto:

```
numeros.sort((a, b) => (b-a));
```

Checa esto en el repositorio GitHub para mas ejemplos de clasificación y tips https://github.com/atapas/js-array-sorting.

El método de array splice()

El método `splice()` ayuda a añadir, actualizar, y remover elementos de un array. Este método puede ser un poco confuso al principio, pero una vez que conoces como usarlo apropiadamente, lo harás bien.

El propósito principal del método `splice()` es eliminar elementos de un array. Regresa un array con los elementos borrados y modifica el array original. Pero puedes añadir y

remplazar elementos usándolo.

Para añadir un elemento usando el método `splice()`, necesitamos pasar la posición donde queremos añadir, cuantos elementos a borrar empezando con la posición, y con el elemento a añadir.

En el ejemplo de abajo, estamos añadiendo un elemento `zack` en el índice `1` sin borrar ningún elemento.

```
const nombres = ['tom', 'alex', 'bob'];

nombres.splice(1, 0, 'zack');

console.log(nombres); // ["tom", "zack", "alex", "bob"]
```

Hecha un vistazo al siguiente ejemplo. Ahí estamos eliminando un elemento del `indice 2` (el tercer elemento) y añadiendo un nuevo elemento, `zack`. El método `splice()` regresa un array con el elemento eliminado, `bob`.

```
const nombres = ['tom', 'alex', 'bob'];

const deleted = nombres.splice(2, 1, 'zack');

console.log(deleted); // ["bob"]
console.log(nombres); // ["tom", "alex", "zack"]
```

Checa esto en este hilo de Twitter y aprende como el método `splice()` te ayuda a hacer un array vacío.

Métodos de arrays estáticos en JavaScript

En JavaScript, tenemos tres métodos de arrays estáticos. Hemos visto ya `Array.isArray()`. Vamos a ver los otros dos ahora.

El método de array Array.from()

Vamos a un fragmento simple de código HTML que contiene un div y una lista de pocos elementos:

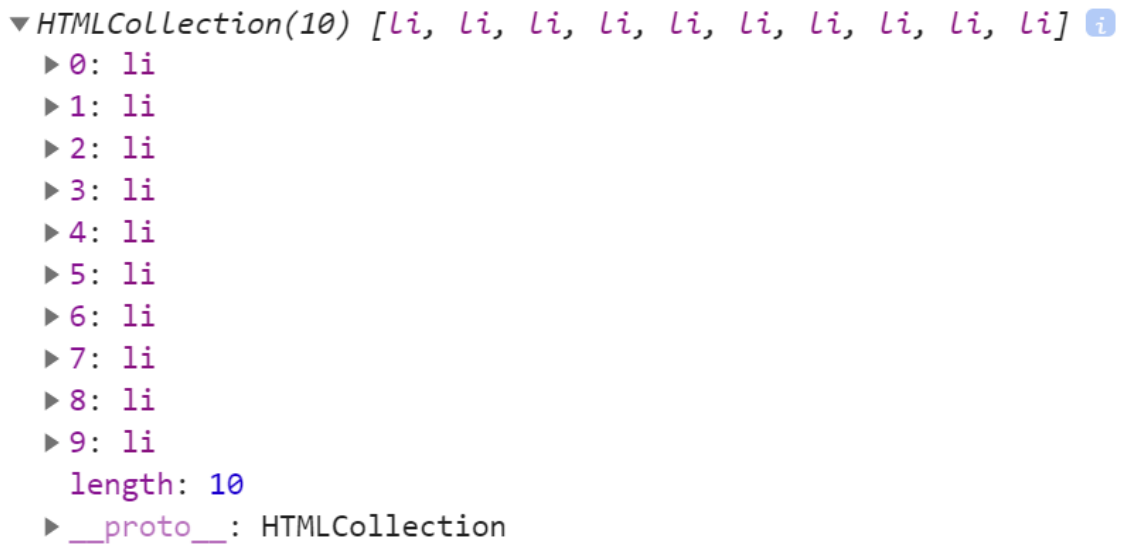
```
<div id="main">
<ul>
```

```
<ol type="1">
<li>...</li>
<li>...</li>
<li>...</li>
<li>...</li>
<li>...</li>
<li>...</li>
<li>...</li>
<li>...</li>
<li>...</li>
<li>...</li>
</ol>
</ul>
</div>
```

Ahora vamos a consultar el DOM usando el método `getElementsByTagName()`.

```
document.getElementsByTagName('li');
```

Regresa un `HTMLCollection` que se ve así:



HTMLCollection is an Array-Like Object

Entonces es como un array. Ahora vamos a tratar de iterar sobre ella usando `forEach`:

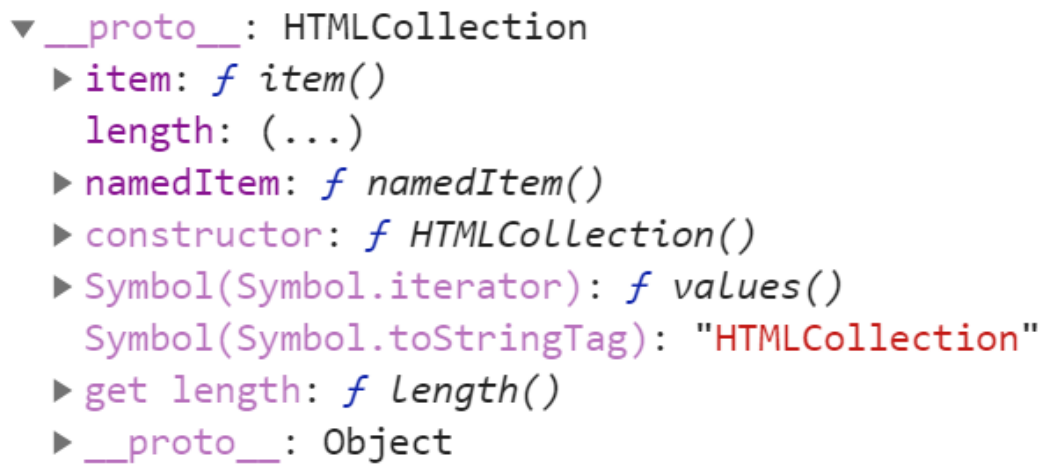
```
document.getElementsByTagName('li').forEach(() => {
// Do something here..
})
```

Adivina ¿cuál es el output? Es un error como este:

```
▶ Uncaught TypeError: document.getElementsByTagName(...).forEach is not a
function
    at <anonymous>:1:37
```

Error while using forEach on the Array-Like object

¿Pero por qué? Por qué la el `HTMLCollection` no es un array. Es un objeto `Array-Like`. Entonces tu no puedes iterar sobre él usando `forEach`.

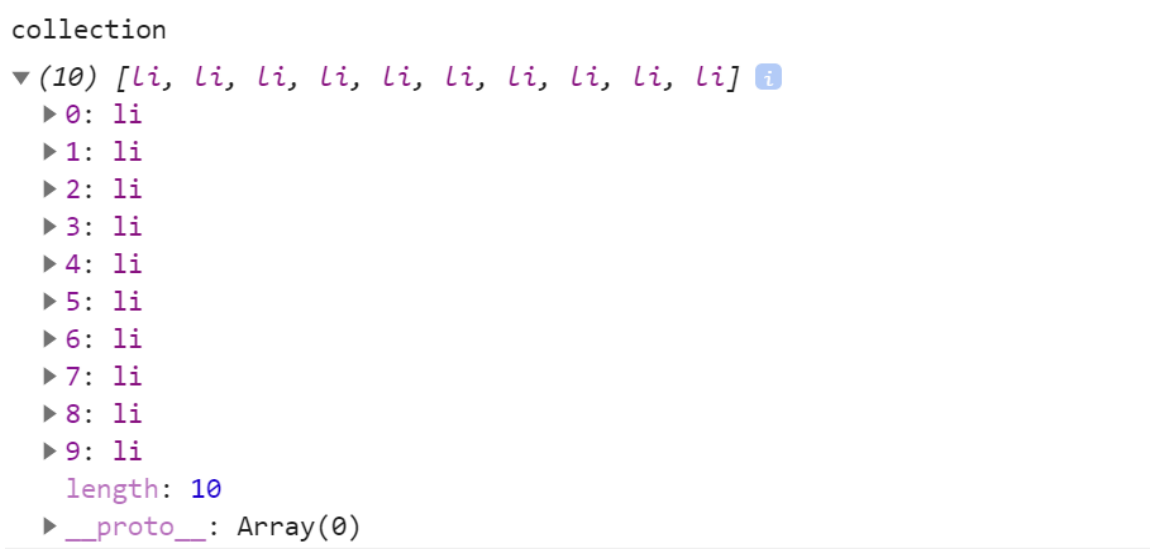


The proto is Object

Aquí es donde deberías usar el método `Array.from()`. Convierte el objeto tipo array a un array para que así puedas realizar todas las operaciones de array en el.

```
const collection = Array.from(document.getElementsByTagName('li'))
```

Aquí esta `collection` que es un array:



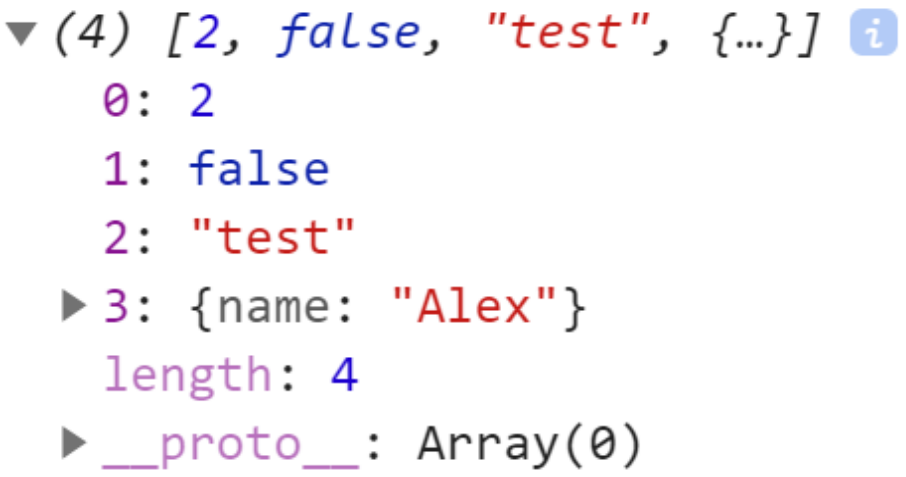
The proto is Array

El método de array Array.of()

El método `Array.of()` crea un nuevo array usando cualquier número de elementos de cualquier tipo.

```
Array.of(2, false, 'test', {'name': 'Alex'})
```

El output se ve como esto:



Output of the Array.of() method

El método de iterador de array en JavaScript

Ahora vamos a aprender acerca de los métodos de iterador de array. Hay mucha métodos muy útiles para iterar a través de un array y realizar cálculos, tomar decisiones, filtrar cosas y más.

Hasta ahora, no hemos visto algún ejemplo de objetos de array. En esta seccion, usaremos los siguientes arrays de objetos para explicar y demostrar los métodos a continuación.

Este array contiene la información de algunos estudiantes suscritos a varios cursos pagados:

```
let estudiantes = [
```

```
{
'id': 001,
'f_nombre': 'Alex',
'l_nombre': 'B',
'genero': 'M',
'casado': false,
'edad': 22,
'paga': 250,
'cursos': ['JavaScript', 'React']
},
{
'id': 002,
'f_nombre': 'Ibrahim',
'l_nombre': 'M',
'genero': 'M',
'casado': true,
'edad': 32,
'paga': 150,
'cursos': ['JavaScript', 'PWA']
},
{
'id': 003,
'f_nombre': 'Rubi',
'l_nombre': 'S',
'genero': 'F',
'casado': false,
'edad': 27,
'pago': 350,
'cursos': ['Blogging', 'React', 'UX']
},
{
'id': 004,
'f_nombre': 'Zack',
'l_nombre': 'F',
'genero': 'M',
'casado': true,
'edad': 36,
'pago': 250,
'cursos': ['Git', 'React', 'Branding']
}
];
```

Bien vamos a empezar. Todos los métodos de array iterativo toma funciones como un argumento. Tú necesitas especificar la lógica para iterar y aplicarla en la función.

El método de array filter()

El método filtró crea un nuevo array con todos los elementos que satisfagan la condición mencionada en la función. Vamos a encontrar el estudiante que es femenino. Entonces la

condición de filtro debería ser que el genero sea igual a 'F'.

```
const femaleStudents = students.filter((element, index) => {
return element.gender === 'F';
})

console.log(femaleStudents);
```

El output es este:

```
▼ [{…}]
  ▶ 0: {id: 3, f_name: "Rubi", l_name: "S", gender: "F", married: false, …}
    length: 1
  ▶ __proto__: Array(0)
```

Eso está bien. El estudiante con el nombre `Rubi` es la única estudiante femenina que tenemos hasta ahora.

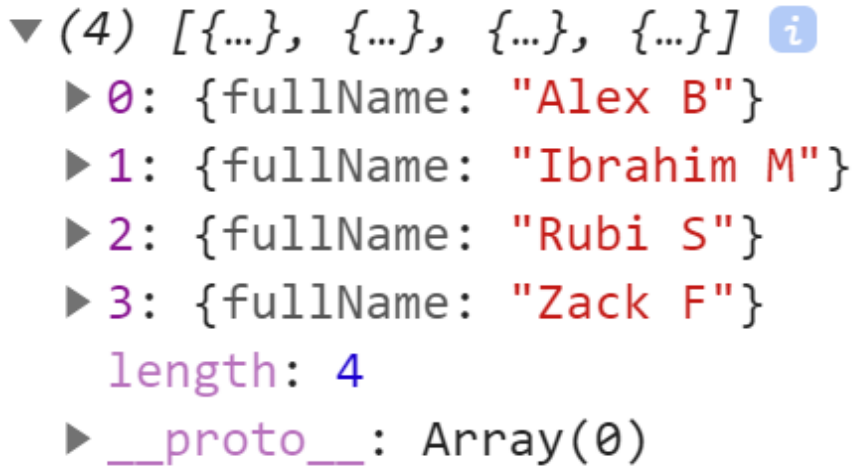
El método de array map()

El método `map()` crea un nuevo array iterando a través de los elementos y aplicando la lógica que proveemos en la función como argumento. Vamos a crear un nuevo array de los nombres completos de todos los estudiantes en el array de `students`.

```
const fullNames = students.map((element, index) => {
return {'fullName': element['f_name'] + ' ' + element['l_name']}
});

console.log(fullNames);
```

El output se veria como esto:



Aquí veremos un nuevo array con el `fullName` propiedades que son calculadas usando las propiedades `f_name` y `l_name` de cada objeto de estudiantes.

El método de array reduce()

El método `reduce()` aplica la función reducer de cada elemento del array y regresa el valor output. Vamos a aplicar la función reducer en el array `students` para calcular la cantidad total a pagar por todos los estudiantes.

```
const total = students.reduce(
(accumulator, student, currentIndex, array) => {
accumulator = accumulator + student.paid;
return (accumulator);
},
0);

console.log(total); // 1000
```

En el código de arriba,

- Iniciamos con el `accumulator` con `0`.
- Aplicamos el método `reduce` en cada uno de los objetos `student`. Accedemos la propiedad `paid` y la añadimos al accumulator.
- Finalmente regresaremos el accumulator.

El método de array some().

El método `some()` regresa un valor booleano (verdadero/falso) basado al menos en un elemento en el array pasando la condición en la función. vamos a ver si ahí hay algún estudiante menor de 30 años de edad.

```
let hasStudentBelow30 = students.some((element, index) => {
return element.age < 30;
});

console.log(hasStudentBelow30); // true
```

Sí, vamos a ver si hay al menos un estudiante menor de 30.

El método de array find()

Usando el metodo `some()`, tenemos que ver si hay algún estudiante debajo de la edad de 30 años. Vamos a encontrar que estudiante es.

Para hacer eso, usaremos el método `find()`. Este regresa el primer elemento encontrado del array que satisface la condición de la función.

Los arrays tienen otro metodo relacionado, `findIndex()`, eso regresa el índice del elemento que encontramos usando el método `find()`. Si no hay elementos que cumplan la condición el `findIndex()` regresa `-1`.

En el ejemplo de abajo, pasamos la función al método `find()` que checa la edad de cada estudiante. Y regresa el estudiante emparejado cuando la condición es cumplida.

```
const estudiante = estudiantes.find((element, index) => {
return element.age < 30;
});

console.log(estudiante);
```

El resultado es este:

```
▼{id: 1, f_name: "Alex", l_name: "B", gender: "M", married: false, …} i
    age: 22
  ▶ courses: (2) ["JavaScript", "React"]
    f_name: "Alex"
    gender: "M"
    id: 1
    l_name: "B"
    married: false
    paid: 250
  ▶ __proto__: Object
```

Como vimos, Alex es quien tiene 22 años de edad. Lo encontramos.

El método de array every()

El método `every()` detecta si cada elemento del array satisface la condición pasada en la función. Vamos a encontrar si todos los estudiantes que se han suscrito al menos dos cursos.

```
const atLeastTwoCourses = students.every((elements, index) => {
return elements.courses.length >= 2;
});
```

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

Como es esperado, podemos ver que el resultado es `true`.

Métodos de arrays propuestos

En mayo de 2021, ECMAScript tiene un método en propuesta, el método at ().

El método at()

El método propuesto `at()` debería ayudarte a acceder a los elementos de un array usando un número de índice negativo. A partir de ahora, esto no es posible. tu puede acceder elemento solo desde el inicio del array usando un número de índice positivo.

Acceder elementos de la parte de atrás del array es posible usando el valor length. Con la inclusión del método `at()`, tú podrás acceder a los elementos usando ambos índices positivo y negativo con un solo método.

```
const junkFoodILove = ['', '', '', '', '', '', '', ''];

junkFoodILove.at(0); //
junkFoodILove.at(3); //
junkFoodILove.at(-1); //
junkFoodILove.at(-5); //
junkFoodILove.at(-8); //
junkFoodILove.at(10); // undefined
```

Aquí hay una demostracion rapida de ello:

Javascript Array at() method demo

Puedes usar este polyfill para lograr la funcionalidad del método `at()` hasta que este método sea añadido al lenguaje JavaScript. Por favor checa este repositorio de GitHub para los ejemplos del método `at()` : https://github.com/atapas/js-array-at-method.

Antes que terminemos...

Espero que te haya resultado útil este artículo, y te ayude a entender los arrays de JavaScript más claramente. Por favor practica los ejemplos multiples veces para obtener un buen retenimiento de ellos. Puedes encontrar todos los ejemplos de codigo en mi repositorio de GitHub.

Conectemonos. Tú puedes encontrarme activo en Twitter (@tapasadhikary). Por favor siente libre de dar follow.