**Metodos base de Array y Recorrer un Array**

* **Tabla de contenido**
* **Métodos pop/push, shift/unshift**
* **Recorrer Arrays**

**Métodos pop/push, shift/unshift**

Antes de continuar con la lectura, mira este video donde se explica detalladamente los conceptos que nos permiten modificar un array. Luego refuerza ese conocimiento con la lectura:

**Array metodos push, pop, shift e unshift**

[**https://www.youtube.com/watch?v=w3RsvedfdYY**](https://www.youtube.com/watch?v=w3RsvedfdYY)

Una cola es uno de los usos más comunes de un array. En ciencias de la computación, significa una colección ordenada de elementos que soportan dos operaciones:

push inserta un elemento al final.

shift obtiene el elemento del principio, avanzando la cola, y así el segundo elemento se vuelve primero.

Los arrays soportan ambas operaciones.

En la práctica los necesitamos muy a menudo. Por ejemplo, una cola de mensajes que necesitamos mostrar en pantalla.

Hay otro caso de uso para los arrays – la estructura de datos llamada pila.

Ella soporta dos operaciones:

push agrega un elemento al final.

pop toma un elemento desde el final.

Entonces los elementos nuevos son agregados o tomados siempre desde el “final”.

Una pila es usualmente mostrada como un mazo de cartas, donde las nuevas cartas son agregadas al tope o tomadas desde el tope:

Para las pilas, la última introducida es la primera en ser recibida, en inglés esto es llamado principio LIFO (Last-In-First-Out, última en entrar primera en salir). Para las colas, tenemos FIFO (First-In-First-Out primera en entrar, primera en salir).

Los arrays en JavaScript pueden trabajar como colas o pilas. Ellos permiten agregar/quitar elementos al/del principio o al/del final.

En ciencias de la computación, la estructura de datos que permite esto se denomina cola de doble extremo o bicola.

Métodos que trabajan sobre el final del array:

pop

Extrae el último elemento del array y lo devuelve:

let fruits = ["Apple", "Orange", "Pear"];

alert( fruits.pop() ); // quita "Pear" y lo muestra en un alert

alert( fruits ); // Apple, Orange

Tanto fruits.pop() como fruits.at(-1) devuelven el último elemento del array, pero fruits.pop() también modifica el array eliminando tal elemento.

push

Agrega el elemento al final del array:

let fruits = ["Apple", "Orange"];

fruits.push("Pear");

alert( fruits ); // Apple, Orange, Pear

El llamado a fruits.push(...) es igual a fruits[fruits.length] = ....

Métodos que trabajan con el principio del array:

shift

Extrae el primer elemento del array y lo devuelve:

let fruits = ["Apple", "Orange", "Pear"];

alert( fruits.shift() ); // quita Apple y lo muestra en un alert

alert( fruits ); // Orange, Pear

unshift

Agrega el elemento al principio del array:

let fruits = ["Orange", "Pear"];

fruits.unshift('Apple');

alert( fruits ); // Apple, Orange, Pear

Los métodos push y unshift pueden agregar múltiples elementos de una vez:

let fruits = ["Apple"];

fruits.push("Orange", "Peach");

fruits.unshift("Pineapple", "Lemon");

// ["Pineapple", "Lemon", "Apple", "Orange", "Peach"]

alert(fruits);

Recorrer Arrays#

Una de las formas más viejas de iterar los items de un array es el bucle for sobre sus índices:

let arr = ["Apple", "Orange", "Pear"];

\*!\*

for (let i = 0; i < arr.length; i++) {

\*/!\*

alert( arr[i] );

}

Pero para los arrays también hay otra forma de bucle,for..of:

let fruits = ["Apple", "Orange", "Plum"];

// itera sobre los elementos del array

for (let fruit of fruits) {

alert( fruit );

}

for..of no da acceso al número del elemento en curso, solamente a su valor, pero en la mayoría de los casos eso es suficiente. Y es más corto.

Técnicamente, y porque los arrays son objetos, es también posible usar for..in:

let arr = ["Apple", "Orange", "Pear"];

\*!\*

for (let key in arr) {

\*/!\*

alert( arr[key] ); // Apple, Orange, Pear

}

Pero es una mala idea. Existen problemas potenciales con esto:

El bucle for..in itera sobre todas las propiedades, no solo las numéricas.

Existen objetos “simil-array” en el navegador y otros ambientes que parecen arrays. Esto es, tienen length y propiedades indexadas, pero pueden también tener propiedades no numéricas y métodos que usualmente no necesitemos. Y el bucle for..in los listará. Entonces si necesitamos trabajar con objetos simil-array, estas propiedades “extras” pueden volverse un problema.

El bucle for..in está optimizado para objetos genéricos, no para arrays, y es de 10 a 100 veces más lento. Por supuesto es aún muy rápido. Una optimización puede que solo sea importante en cuellos de botella, pero necesitamos ser concientes de la diferencia.

En general, no deberíamos usar for..in en arrays.

Recorrer Array

**Recorrer array con un bucle for**

[**https://www.youtube.com/watch?v=iSi\_DY4zzkc**](https://www.youtube.com/watch?v=iSi_DY4zzkc)