**React**

**Arreglos.**

Documentación: <https://es.reactjs.org/>

<https://es.reactjs.org/docs/getting-started.html>

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

Un arreglo es una colección de información que se encuentra dentro de una misma variable.

Los arreglos son objetos similares a una lista cuyo prototipo proporciona métodos para efectuar operaciones de recorrido y de mutación.

Tanto la longitud como el tipo de los elementos de un arreglo son variantes. Dado que la longitud de un arreglo puede cambiar en cualquier momento, y los datos se pueden almacenar en ubicaciones no contiguas, no hay garantía de que los arreglos de JavaScript sean densos. Ejemplo:

const numeros = [1, 2, 3, 4, 5];

/\*

    Al hacer push en el arreglo, JS no marca error en la constante,

    por qué no estamos manipulando a lo que apunta en memoria la constante.

    Sino que estamos utilizando un método propio del arreglo,

    una función que se encuentra dentro del prototipo de los arreglos.

\*/

numeros.push(6);

/\*

    Caso contrario aquí, JS marcará el error en la constante,

    ya que estamos tratando de reasignar el valor de la constante.

\*/

numeros = [1, 2, 3, 4, 5, 6];

console.log(numeros);

**Nota importante.**

**En los arreglos pasa lo mismo que con los objetos, los arreglos son básicamente un objeto por lo cual también son pasados por referencias, esto quiere decir que cualquier modificación que sufra el arreglo inicial, las demás referencias que se hicieron al arreglo inicial se verán afectadas de igual manera, ya que todas las referencias están apuntando al mismo lugar. Ejemplo:**

const numeros = [1, 2, 3, 4, 5];

numeros.push(6);

const masNumeros = numeros;

masNumeros.push(7); // Este cambio en masNumeros, afectará también al arreglo de numeros.

console.log(numeros);

console.log(masNumeros);

**Para poder romper esa referencia del arreglo inicial y hacer que no se modifiquen los cambios en el arreglo inicial, una manera muy sencilla es utilizar algo conocido como el operador SPREAD, este operador se especifica con los “…” a la izquierda del arreglo que se hará la referencia, esto esparcirá cada una de los valores del arreglo al que se indica, sin apuntar a su referencia. Mejor dicho este operador crea un nuevo arreglo y rompe la referencia del otro arreglo. Esto es una sintaxis bastante común hoy en día en JS.**

[**https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/Spread\_syntax**](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/Spread_syntax)

**Ejemplo:**

const numeros = [1, 2, 3, 4, 5];

numeros.push(6);

const masNumeros = numeros;

masNumeros.push(7); // Este cambio en masNumeros, afectará también al arreglo de numeros.

const otrosNumeros = [...numeros];

otrosNumeros.push(8); // Este cambio en otrosNumeros, no afectará al arreglo de numeros y masNumeros. gracias al operador REST.

console.log(numeros);

console.log(masNumeros);

console.log(otrosNumeros);

**Existen otro operador que se llamado REST y la diferencia está dependiendo de donde se utilice con el arreglo.**

[**https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Functions/rest\_parameters**](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Functions/rest_parameters)

En los arreglos no siempre es necesario utilizar el operador SPREAD, hay funciones o métodos dentro de los arreglos que nos pueden ayudar a regresar un numero arreglo, ese nuevo arreglo ya rompería la referencia al arreglo inicial. Ejemplo:

const numeros = [1, 2, 3, 4, 5, 6, 7];

// La función map de un arreglo llama a una función de devolución de llamada definida en cada uno de los elementos de un arreglo y devuelve un arreglo que contiene los resultados.

// En pocas palabras la función map recorre/itera cada uno de los elementos dentro de un arreglo y cambia el valor de los elementos dependiendo lo que se le indique

const multiplicaNumeros = numeros.map( (numero) => numero \* 2);

multiplicaNumeros.push(100);

console.log(numeros); // [1, 2, 3, 4, 5, 6, 7]

console.log(multiplicaNumeros); // [2, 4, 6, 8, 10, 12, 14, 100]