La función map()

Muchas veces necesitamos realizar una operación por cada elemento de un array y la solución pasa por realizar un bucle entre todos los elementos del array e ir recuperando y ejecutando cierto código por cada elemento. ¿Sería posible hacer esto de una forma sencilla? La respuesta es sí, mediante la función map().
A continuación se muestra un código sencillo explicado paso a paso:

```
<!DOCTYPE html>
<html>
<body>

<p id="resultado">Tenemos el array definido de la siguiente manera:
let numeros = [1, 2, 3, 4, 5, 6, 7]; cuando pulses el botón aquí
aparecerán los mismos números incrementados en uno</p>

<button onclick="dale()">ejecuta</button>

<script>
let numeros = [1, 2, 3, 4, 5, 6, 7];

function dale() {
 let obj = document.getElementById("resultado")
 obj.innerHTML = numeros.map(function(valor){return valor+1;});
 obj.innerHTML = numeros.map((v,i)=>v+i);
 obj.innerHTML = numeros.map((v,i)=>{return(v+i)});


 /* map irá ejecutando la función incluida como parámetro la cual no
tiene nombre porque no es necesario. Esta función tomará como parámetro
valor (el valor de esa posición del array) y devolverá dicho valor
aumentado en uno. La función map devuelve otro array con el resultado
de aplicar la función anterior a los elementos de todo el array */
}
</script>

</body>
</html>
```

Recuerda
La función map devuelve otro array con el resultado de aplicar la función pasada como

parámetro a los elementos del array.
Map es la función SIN (sin bucles y sin tener que crear un array puesto que lo hace ella sola).

El prototipo de esta función map() es el siguiente:

```
array.map(function(elemento, indice, el_array), valor_this)
```

La función map tiene dos parámetros como se puede observar. El primero es la **función callback** que a su vez tiene tres parámetros, y el segundo parámetro es valor_this el cual es opcional y se podrá utilizar como “this” para referenciar al objeto de la llamada.
En la función callback el único parámetro obligatorio “es elemento” (el primero). “Índice” que indica el índice del elemento, y “el_array” que apunta al array protagonista de la llamada son opcionales.

Si queremos realizar un código que tenga en cuenta el elemento y su índice, un ejemplo podría ser el siguiente:

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

function dale() {
 let obj = document.getElementById("resultado")
 obj.innerHTML = numeros.map(function(valor,indice){return
valor+indice;});
 /* el resultado de esta llamada será [1,3,5,7,9,11,13] */
}
```

El anterior código es una modificación del programa ejemplo primero y su función es sumar a cada elemento del array el índice que ocupa en dicho array.

Importante
No hace falta que utilices la función map() obligatoriamente para devolver un array.
Puedes utilizarla para realizar una operación por cada elemento del array.

```
let numeros = [1, 2, 3, 4, 5, 6, 7];
numeros.map(function(valor,indice){console.log("valor:"+valor+"
indice:"+indice);});
```

Como se puede ver en el código anterior, se mostrará por consola el valor de los elementos del array y su índice.

La función arrow o función flecha =>

Muchas ocasiones lo más normal es encontrarnos con la función map() en combinación con la función flecha => o función anónima. Veamos dos líneas de código funcionalmente iguales:

```
 obj.innerHTML = numeros.map(function(valor,indice){return
valor+indice;});
 obj.innerHTML = numeros.map((valor,indice)=>{return valor+indice;});
```

Como se puede observar en el código anterior se ha sustituido la palabra reservada function por su equivalente inline que es la función arrow "=>".
La función flecha se utiliza para escribir funciones callback ad-hoc y es muy común su utilización en muchos frameworks como **ReactJS** o **Angular**. Como muchos de estos framework utilizan muy a menudo funciones **callback**. La función arrow es una forma intuitiva, fácil y rápida de programar dichos callback.

La función reduce()

La función reduce permite iterar un array acumulando las operaciones que se vayan haciendo sobre los elementos en otro llamado acumulador. A continuación se ponen dos ejemplos de uso:

```
let numeros = [1, 2, 3, 4];
console.log (numeros.reduce((acc,v)=>{return acc+v;}));
let numeros2 = ["uno","dos","tres","cuatro"];
console.log (numeros2.reduce((a,b)=>{return a.concat(b);}));
```

En el primero de ellos se sumarán todos los números del array (1+2+3+4) lo cual mostrará por consola en la segunda línea el valor 10.
El segundo ejemplo, mostrará por consola la concatenación de todos y cada uno de los elementos del array. El resultado será: "unodostrescuatro".

En el siguiente esquema se puede ver tanto el prototipo de la función reduce() como su aplicación al primer ejemplo del código anterior:

Ejemplo:

```
let numeros = [1, 2, 3, 4];
console.log (numeros.reduce((acc,v)=>{return acc+v;}));
```

Como se puede observar, reduce también utiliza funciones callback y por lo tanto la función arrow (=>) vista anteriormente es una candidata perfecta a utilizar.

Recuerda
Utiliza la función map() cuando tengas que realizar una operación por cada elemento de un array.
Utiliza la función reduce() cuando necesites realizar una operación por cada elemento de un array y haya que ir acumulando el resultado generado.
Utiliza la función arrow (=>) para funciones callback.