Hola, en esta entrada vamos a explicar un poco los métodos nativos map, filter y reduce, que encontramos en los objetos Arrays de JavaScript y como pueden ayudar a hacer más eficiente la implementación, ya que son más rápidos que los bucles tradicionales.

- map([□, □, □], cook) => [□, □, □]
- filter([□, □, □], isVegetarian) => [□, □]
- reduce([□, □], eat) => □

Map

El método map() devuelve un nuevo Array como resultado de la función que se está aplicando a cada elemento del Array. Esto significa que cada elemento del Array va a ser devuelto en la misma posición del Array resultante pasando por la función del método map(), como lo haría un iterador.

```
// Declaramos un Array
> var a = new Array(1, 2, 3) || [1, 2, 3]
// Mapeamos el Array con una función para elevar al cuadrado el contenido
> var resultado = a.map(elemento => Math.pow(elemento, 2))
// Se imprime el resultado
> console.log(resultado)
> [1, 4, 9]
```

Finalmente el método map() devuelve un nuevo Array de la misma dimensión que el original pero modificando cada elemento por el de la salida de la función que estemos aplicando, por lo que no se modifica el Array original.

Filter

El método filter() del objeto Array, como su nombre indica se utiliza para filtrar los elementos del Array con las condiciones lógicas que se tenga en la función que implementa, si la condición no pasa el filtro este elemento no es devuelto en el Array resultante, quedando una dimensión menor que el original.

```
// Declaramos un Array
> var a = new Array(1, 2, 3) || [1, 2, 3]
// Filtramos el Array con una función para obviar el número 2
> var resultado = a.filter(elemento => elemento !== 2)
// Se imprime el resultado
> console.log(resultado)
> [1, 3]
```

Vemos como el filtro ignora el número 2 ya que la condición es que sea distinto del número 2, por lo que este número si se encuentra en el Array no es devuelto en el resultante y su posición se obvia, tampoco se modifica el Array original.

Reduce

El método reduce() del objeto Array, este método reduce el Array a un único elemento, ejecuta la función que tiene el método en cada uno de los elementos del Array y el resultado de cada elemento se almacena en un acumulador que será el resultado total.

```
// Declaramos un Array
> var a = new Array(1, 2, 3) || [1, 2, 3]
// Reducimos el Array a un solo elemento sumando los cuadrados de los elementos del Array
> var resultado = a.reduce((acumulador, elemento) => acumulador + Math.pow(elemento,2))
// Se imprime el resultado
> console.log(resultado)
> 14
```

Como vemos el resultado en este caso no es un nuevo Array sino el acumulador de la función que apliquemos, tampoco se modifica el Array original.

Conclusiones

Para terminar añadir que ninguno de estos tres métodos utiliza un elemento que no esté definido, en caso de que sea undefined o null se obvia la función aplicada.

Es una buena práctica utilizar los métodos nativos de los Arrays en vez de los *bucles*

tradicionales, ya que son mucho más rápidos a la hora de iterar y si estamos aplicando funciones se hace todo con una sola instrucción y es más limpio.