Contenido

En Javascript , se introducen unos métodos muy útiles para utilizar como **iteradores**.
Un **iterador** es un concepto que se repite, y en el ámbito de la programación, se suele referir a algo que te permite recorrer una estructura de datos por todos sus apartados o miembros. 1

En Javascript , se introducen unos métodos muy útiles para utilizar como **iteradores**.
Un **iterador** es un concepto que se repite, y en el ámbito de la programación, se suele referir a algo que te permite recorrer una estructura de datos por todos sus apartados o miembros.

En muchos casos, se presenta la situación en la que necesitamos **recorrer un objeto**, a través de las propiedades de su estructura, como si fueran los elementos de un array. Sin embargo, al ser un objeto parece que no es posible.

ITERADORES DE OBJETOS

Sin embargo, existen unos métodos denominados **Object.keys()**, **Object.values()** y **Object.entries()** que nos van a permitir realizar esta tarea. En primer lugar, observa que son métodos de una Clase estática, por lo que tienes que escribir siempre el **Object.** y no ejecutar el método sobre el objeto en sí, como solemos hacerlo.

Veamos de que métodos se trata:

Método	Descripción
Object.keys(obj)	Itera el **obj** y devuelve sus propiedades o **keys**.
Object.values(obj)	Itera el **obj** y devuelve los **valores** de sus propiedades.
Object.entries(obj)	Itera el **obj** y devuelve un con los pares **[key, valor]**.
Object.fromEntries(array)	Construye un objeto con un array de pares **[key, valor]**.

En cualquiera de los 3 casos, devuelven siempre un array.

Convertir un objeto a array

Todo esto puede parecer complejo, pero en realidad es muy sencillo. Veamos un ejemplo para entender como funcionan:

```
const user = {
 name: "Manz",
 life: 99,
 power: 10,
 talk: function() {
  return "Hola!";
 }
};

Object.keys(user);      // ["name", "life", "power", "talk"]
Object.values(user);    // ["Manz", 99, 10, f]
Object.entries(user);   // [["name", "Manz"], ["life", 99], ["power", 10], ["talk", f]]
```

Expliquemos este código:

- Con el método **Object.keys()** obtenemos un arrayde las claves (*propiedades, índices, keys*) del objeto.
- Con el método **Object.values()** obtenemos un array de los valores de las claves anteriores, en el mismo orden.
- Con el método **Object.entries()** obtenemos un array de **entradas**.
  Cada **entrada** es un del par clave-valor, es decir, la propiedad del objeto original y su valor correspondiente.

Ten en cuenta que como un array también es un objetct , podemos utilizar estos métodos también para recorrerlos, sólo que en este caso los índices del array son las posiciones (*0, 1, 2, 3...*). Veamos un ejemplo:

```
const animals = ["Gato", "Perro", "Burro", "Gallo", "Ratón"];

Object.keys(animals);   // [0, 1, 2, 3, 4]
Object.values(animals); // ["Gato", "Perro", "Burro", "Gallo", "Ratón"]
Object.entries(animals); // [[0, "Gato"], [1, "Perro"], [2, "Burro"], [3, "Gallo"], [4,
"Ratón"]]
```

Cualquier otra estructura se podría utilizar con **Object.keys()**, **Object.values()** o **Object.entries()** pero devolverá un array vacío si no se puede iterar. Por ejemplo:

```
Object.keys(new Date());  // []  (Iterar una fecha)
Object.values(/.+/);      // []  (Iterar una expresión regular)
Object.entries(true);     // []  (Iterar un booleano)
```

Así pues, los casos más interesantes suelen ser estructuras de object y, quizás en algunos casos, de arrays .

Convertir un array a objeto

De la misma forma, también se puede hacer la operación inversa. Para ello, usaremos el método **Object.fromEntries()**. En esta ocasión, vamos a partir de dos arrays **keys** y **values**, donde el primero tiene la lista de propiedades en string y el segundo tiene la lista de valores.

El objetivo es, a partir de esos dos arrays (*que deben ser del mismo tamaño*), generar el objeto inicial **user** que teníamos antes:

```
const keys = ["name", "life", "power", "talk"];
const values = ["Manz", 99, 10, function() { return "Hola" }];

const entries = [];
for (let i of Object.keys(keys())) {
  const key = keys[i];
  const value = values[i];
  entries.push([key, value]);
}

const user = Object.fromEntries(entries);    // {name: 'Manz', life: 99, power: 10, talk:
f}
```

Observa que partimos de un array vacío **entries**. Con **Object.keys(keys)** obtenemos una lista de números de **0** al tamaño del array **keys**. Esto nos servirá de posición para ir recorriendo los arrays **keys** y **values** en el interior del bucle **for..of**.

De esta forma, en cada iteración del bucle generamos un par **key**, **value**, que meteremos en un array e insertaremos en **entries**. De esta forma, regeneramos la estructura de entradas de **Object.entries()** que es la que necesitamos para que, mediante **Object.fromEntries()** podamos regenerar el objeto **user** con las keys de **keys** y los valores de **values**.

Otra forma, más compacta, pero que quizás requiere más experiencia, sería la siguiente:

```
const keys = ["name", "life", "power", "talk"];
const values = ["Manz", 99, 10, function() { return "Hola" }];

const entries = values.map((value, index) => [keys[index], value]);
const user = Object.fromEntries(entries);
```

En este caso hacemos exactamente lo mismo, pero utilizamos el método **.map()** del tema de las array functions para simplificar el bucle.