ThePower business school

🇪🇸 Bucles JS 🔁🤖

Los bucles son una herramienta muy útil en JavaScript y en cualquier otro lenguaje de programación. Sirven para ejecutar una acción o conjunto de acciones varias veces, según una condición o una serie de condiciones.

Hay varios tipos de bucles en JavaScript, y cada uno tiene su propia sintaxis y usos específicos. A continuación, te explicaré brevemente cada uno de ellos y te proporcionaré algunos ejemplos de su uso.

Bucle `for`

El bucle `for` es uno de los más comunes en JavaScript y se utiliza para ejecutar un bloque de código un número determinado de veces. La sintaxis de un bucle `for` es la siguiente:

```
for (inicialización; condición; actualización) {
  // código a ejecutar
}
```

La inicialización se ejecuta una vez al principio del bucle y suele utilizarse para inicializar una variable de control. La condición se evalúa al principio de cada iteración del bucle y, si se cumple, se ejecuta el bloque de código. Si la condición no se cumple, el bucle termina. Por último, la actualización se ejecuta al final de cada iteración del bucle y suele utilizarse para modificar la variable de control.

Aquí tienes un ejemplo de un bucle `for` que imprime los números del 1 al 10:

```
for (let i = 1; i <= 10; i++) {
  console.log(i);
}
```

En este caso, la variable `i` se inicializa a 1 al principio del bucle. Luego, se evalúa la condición `i <= 10`, que se cumple mientras `i` sea menor o igual que 10. Por tanto, se ejecuta el bloque de código, que imprime el valor de `i` en la consola. Al final de cada iteración, se ejecuta la actualización `i++`, que incrementa el valor de `i` en 1.

Bucle `while`

El bucle `while` es otro tipo de bucle muy común en JavaScript. Se utiliza para ejecutar un bloque de código mientras se cumpla una determinada condición. La sintaxis de un bucle `while` es la siguiente:

```
while (condición) {
  // código a ejecutar
}
```

Aquí tienes un ejemplo de un bucle `while` que imprime los números del 1 al 10:

```
let i = 1;
```

Bucle `for`

Es el bucle más básico y funcional. se inicia con la palabra reservada **for** y a continuación la condición de iteración. Es decir entre paréntesis lleva la inicialización **let** `index = 0`, la condición de ruptura `index < array.length` y el incremento `index ++` o decremento `index --`

```
for (let index = 0; index < array.length; index++) {
  let element = array[];
}
```

Vamos a ver un ejemplo más practico de un bucle y cómo usarlo, ya que hasta ahora no lo hemos visto en acción. Creamos una lista de números y lo iteramos en cuanto se cumpla la condición queremos sacar un mensaje por consola.

```
let numberList = [1,2,3,4,7,8,10,13];

for (let index = 0; index < numberList.length; index++) {
  if(numberList[index] === 13) {
    console.log('Dicen que da mala suerte 👹')
  }
}
```

Podemos anidar tantos bucles como queramos, es decir si nuestro elemento iterado tiene a su vez otro elemento que es posible iterar podemos usar un bucle dentro de otro bucle.

```
let studentList = [
  {
    name: 'JdelaCruz',
    codeList: ['Js', 'React']
  },
  {
    name: 'EdDarko',
    codeList: ['Js', 'Node']
  }
]

for (let i = 0; i < studentList.length; i++) {
  for (let j = 0; j < studentList[i].codeList.length; j++) {
    console.log(studentList[i].name + 'Works: ' + studentList[i].codeList[j])
  }
}
```

Bucle `foreach`

Podemos construir un bucle **for**, pero con una estructura más funcional gracias al **foreach**. En ocasiones puede ser muy útil generar una función para ejecutarla por cada vuelta de bucle.

```
let studentList = ['AlbaNav', 'JdelaCruz', 'EdDarko', 'Corocotax'];

/* No arrow */
studentList.forEach(
  function (element) {
    console.log(element);
  }
);

/* arrow */
studentList.forEach(element => console.log(element))

/* Otro ejemplo */
let printNames = (element) => {
  console.log(element);
}

studentList.forEach(printNames);
```

Bucle `for of`

ES6 nos permite ejecutar un atajo para realizar bucles llamado **for..of**. El bucle **for...of** automáticamente nos devuelve los **valores** de los elementos actuales iterando a través del objeto a través de la siguiente sintaxis.

```
for (variable of objeto_iterable) { ... }
```

Este bucle itera a lo largo de objetos iterables como **Array**, **Map**, **String**, **Arguments**, etc...

```
// Array

let ninjaTurtles = ['Leonardo', 'Michelangelo', 'Donatello', 'Raphael'];

findLongestWord

// String

let name = 'Master Khan';
for (var word of name) {
  console.log(word);
```

```
}

// Set
let powerRangers = new Set();
powerRangers.add(1);
powerRangers.add("Rojo");

for (var ranger of powerRangers) {
    console.log(ranger);
}

// Arguments

function mixArgs() {
  for (var value of arguments) {
    console.log(value);
  }
}
crazyContainer(ninjaTurtles, name);
```

Bucle `for in`

El método **for...in** nos da la posibilidad a la hora de generar bucles de recorrer todos los índices del objeto, de esta manera no obtendremos el valor, sino la propiedad enumerable.

Por ejemplo, en el caso de un array obtendremos la posición de cada uno de los elementos y en el caso de un objeto obtendremos la clave.

```
// Objeto
let batman= {
    nombre: "Bruce",
    apellidos: "Wayne",
    localizacion: "Gotham",
    profesion: "Multimillonario"
}

for (let clave in batman) {
    console.log("Batman tiene la clave " + clave + " con valor: " + batman[clave]);
}

//return
//Batman tiene la clave nombre con valor: Bruce
//Batman tiene la clave apellidos con valor: Wayne
//Batman tiene la clave localizacion con valor: Gotham
```

```
//Batman tiene la clave profesion con valor: Multimillonario

// Array

const eternals = [
  "Ikaris",
  "Kingo",
  "Sersi",
  "Sprite",
  "Phastos",
  "Makkari",
  "Druig",
  "Gilgamesh",
];

for (const eternal in eternals) {
  console.log(
    "Este Eterno tiene la posición " + eternal + " con el valor: " + eternals[eternal]
  );
}

//return
//Este Eterno tiene la posición 0 con el valor: Ikaris
//Este Eterno tiene la posición 1 con el valor: Kingo
//Este Eterno tiene la posición 2 con el valor: Sersi
//Este Eterno tiene la posición 3 con el valor: Sprite
//Este Eterno tiene la posición 4 con el valor: Phastos
//Este Eterno tiene la posición 5 con el valor: Makkari
//Este Eterno tiene la posición 6 con el valor: Druig
//Este Eterno tiene la posición 7 con el valor: Gilgamesh
```

`for of` VS. `for in`

Una de las diferencias es que **for-of** solamente puede iterar en objetos iterables, en cambio, **for-in** puede iterar en cualquier tipo de objeto. Otra diferencia, es que **for-in** devuelve las claves y **for-of** los valores.

Vamos a hacer una prueba del uso de **for-of** y de **for-in** seguro que os sorprende los resultados.

```
var dieHardArray = [1, 2,'Simon', 'John McClane', 'Zeus Carver'];

var dieHardObj = {
  name: 'John',
  surname: 'McClane',
```

```
  age: 37
};

// Iterar un Array
for (value of dieHardArray) {
  console.log(value);
}

for (key in dieHardArray) {
  console.log(key);
}


// Iterar un Objeto

for (key in dieHardObj) {
  console.log(key);
}

for (value of dieHardObj) {
  console.log(value);
}
```

`for of` vs. `for each`

La principal diferencia es que **for-of** puede iterar en cualquier tipo de objeto iterable, en cambio, **forEach** solamente puede en arrays. Lo vemos con un ejemplo.

```
var backToTheFutureArray = [21, 10, 2015, 'Delorean'];
var backToTheFutureString = 'Dr.Emmett Brown';

// Iterar un Array
for (value of backToTheFutureArray) {
  console.log(value);
}

backToTheFutureArray.forEach(function(value, index) {
  // podemos acceder al índice
  console.log(value, index);
});


// Iterar un String
for (value of backToTheFutureString) {
  console.log(value);
}

backToTheFutureString.forEach(function(value, index) {
```

```
  console.log(value, index);
});
```