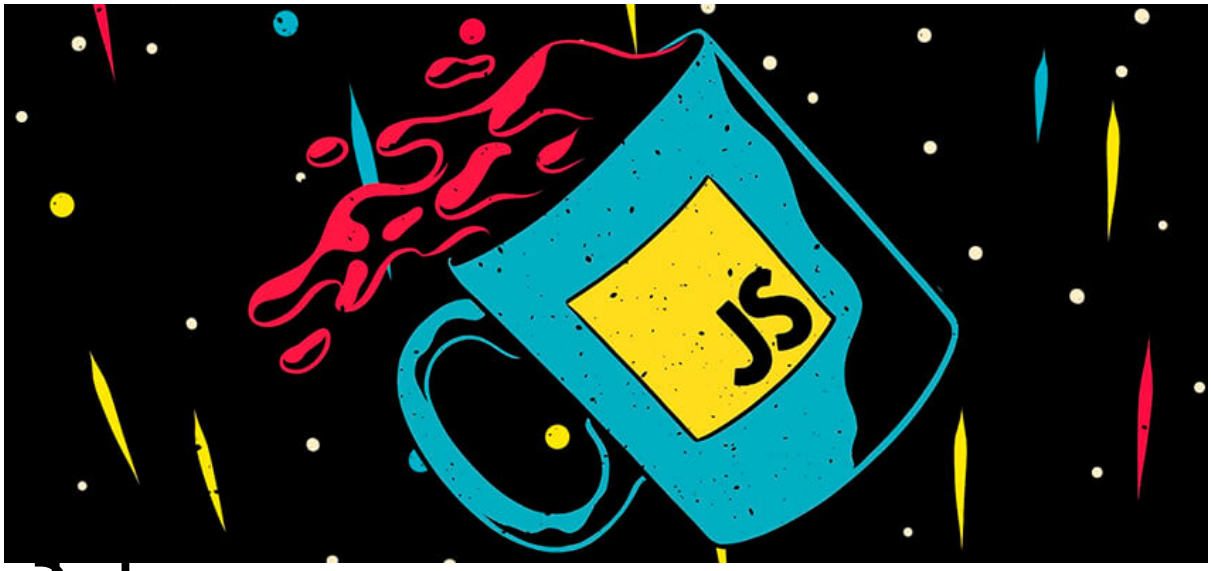

Javascript S3 | Condicionales - Bucles

Después de esta lección podrás:

1. Trabajar con condicionales avanzados.
2. Usar condiciones ternarias dejando un código más limpio.
3. Trabajar con bucles avanzados.
4. Aprender a utilizar de un bucle u otro en función de la necesidad.

Los Condicionales

Los condicionales son estructuras de control de JavaScript que sirven para ejecutar un código u otro (o ninguno) en función de si se cumple o no una condición. En ellos se establece una **condición** y el código en caso de que se cumpla o no, si esa condición se cumple se ejecuta un código y sino otro o ninguno. ***Si esta condición es verdadera, haz esto y sino esto otro***. La condición que escribamos siempre se va a convertir en `true` o `false`.

If:Podemos pensar en ellos como un *"Si...haz..."*.

```
var age = 35;
if (age > 30) {
  alert('Tienes más de 30 años');
//Esta línea se ejecuta solo si se cumple la condición
}
```

Existe otra estructura para el condicional cuando queremos que ejecutar un código diferente cuando no se cumpla la condición. Partiendo de la estructura simple, añadimos:

Else:Podemos pensar en ello como un *"Si...haz...sino haz..."*.

```
var age = 35;
if (age > 30) {
  alert('Tienes más de 30 años');
//Esta línea se ejecuta solo si se cumple la condición
} else {
  alert('Como mucho tienes 30 años');
//Esta línea se ejecuta solo si NO se cumple la condición
}
```

Si necesitamos una estructura más complicada, siempre podemos poner un `else` al final para ejecutar código cuando no se ha cumplido ninguna de las condiciones. Además, podemos incluir todas las condiciones que queramos con `else if`.

Podemos pensar en ello como un *"Si...haz...sino si...haz..."*.

```
let mutant = 'wolverine';
if (mutant == 'ciclops') {
  alert('Who are you?');
  // Esta línea se ejecuta solo si se cumple la condición
```

```
} else if (mutant == 'beast') {
  alert('Beast? Hank?');
// Esta línea se ejecuta solo si se NO cumple la primera condición
// y SÍ se cumple la segunda
}
```

También tenemos una situación similar a la del if...else pero en este caso en función del valor que entra dentro del **switch** ejecutará un case u otro. El break nos permite que cuando encuentre su caso no siga ejecutando el **switch**:

S**witch:** Podemos pensar en ello como un *"Si...coincide haz ...sino coincide nunca default ..."*.

```
var name  = prompt ("Favorite Marvel main character:");
var team = "";

switch (name) {
  case "Daredevil":
    team = "The Defenders";
    break;
 case "Spiderman":
    team = "Avengers";
    break;
 case "Black bolt":
    team = "Inhumans";
    break;
 case "Beast":
    team = "X-Men";
    break;
 default:
    team = "Team Marvel"
    break;
}

console.log("Your favorite character is from the team " + team);
```

El operador ternario de JavaScript (y en otros lenguajes de programación) es una forma abreviada de la sentencia ***if else*** para tomar una decisión, usarla nos ayuda a crear código más limpio y fácil de entender y además nos ayuda a escribir código más rápido por que hay menos caracteres que escribir.

Se llama ternario porque consta de 3 partes, la primera es la condición, la segunda el valor que regresa si la condición es verdadera y el tercero es el valor que retorna si la condición es false.

Ambos valores para falso y verdadero se separan entre ellos con un signo `:` mientras en el signo `?` se usa para separar la condición de los posibles valores `falso` y `verdadero`. Muy sencillo.

Para entenderlo veamos primero la versión larga con un ejemplo en el que vamos a mostrar un mensaje en pantalla diciendo si alguien aprueba o falla un examen basándonos en una puntuación:

```
let score = 5;

if(score >= 6) {
alert('aprobado');
} else {
alert('suspenso')
}
```

Ahora veamos la versión con ternario:

```
let score = 7;
alert ( score >= 6 ? 'aprobado' : 'suspenso' );
```

Se ve mucho mejor ¿cierto?, pero ahora voy a explicarte todo esto de forma muy sencilla, el operador ternario trabaja de esta forma:

```
score >= 6 //condición
? // si
'aprobado' // verdadero
```

```
: // or
'suspenso' // falso
```

Los Bucles

Foreach - Cuando sacaron ES6 incluyeron una nueva forma de recorrer los elementos de un array, el bucle **foreach**:

```
myArray.forEach(function (value) { console.log(value);});
```

Este bucle se suele usar en favor del bucle for cuando necesitamos encapsular código ya que pasamos una función como parámetro al bucle foreach. Esta función se puede sustituir por una **arrow function**:

```
myArray.forEach((value) => { console.log(value);});
```

La ventaja de este tipo de bucles es que el código es mucho más mantenible ya que no tienes que declarar contadores todo el rato, sobre todo si es para recorrer arrays y listas.

Por cierto, si quieres usar un contador dentro del foreach lo puedes hacer así:

```
myArray.forEach((value, i) => { console.log(value);});
```

La variable **i** se incrementará automáticamente en cada iteracción del bucle.

For-of -El bucle **for-of** es un nuevo tipo de bucle introducido en ES6. Este bucle lo que nos permite es iterar a través de los **elementos de objetos iterables** como, por ejemplo, **String, Array, Set, Map,** etc.

```
for (value of iterable_obj) { ...}
```

Vamos a ver unos ejemplos con algunos de los tipos de objetos que hemos comentado antes que permite iterar:

```
// ARRAY
const array = [1, 2, 'a'];
for (let value of array) {
  console.log(value);
}
// Resultado:
// 1
// 2
// a


// STRING
const string = 'string';
for (let value of string) {
  console.log(value);
}
// Resultado:
// s
// t
// r
// i
// n
// g


// ARGUMENTS
function testFunction() {
  for (let value of arguments) {
    console.log(value);
  }
}
testFunction(1, 2, 'a', [string]);
// Resultado:
// 1
// 2
// a
// [ 'string' ]
```

For-in - Para poder recorrer los índices de un objeto JavaScript nos ofrece la función **for in**: Mediante la función for in podemos recorrer todos los índices del objeto, de manera que podemos ir accediendo a cada una de sus propiedades.

```
// Definimos un objeto con los datos de una persona
var persona = {
nombre: "Peter",
apellidos: "Parker",
pais: "USA",
profesion: "Student"
}

for (var indice in persona) {
ºconsole.log("Persona tiene " + índice + " con valor: " + persona[indice]);
}
```

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.

```
const array = [3, 4, 5, 'string'];
const obj = {
  name: 'John',
  age: 20
};

// ARRAY
for (value of array) {
  console.log(value);
}
// Resultado:
// 3
// 4
// 5
// string
```

```
for (key in array) {
  console.log(key);
}
// Resultado:
// 0
// 1
// 2
// 3

// OBJETO
for (key in obj) {
  console.log(key);
}
// Resultado:
// name
// age

for (value of obj) {
  console.log(value);
}
// Resultado:
// TypeError: obj[Symbol.iterator] is not a function
```

La principal diferencia es que **for-of** puede iterar en cualquier tipo de objeto iterable, en cambio, **.forEach** solamente puede en arrays. Otra diferencia es que con **.forEach** podemos acceder al índice y con **for-of** no.

```
const array = [3, 4, 5, 'string'];
const string = 'string';

// ARRAY
for (value of array) {
  console.log(value);
}
// Resultado:
// 3
// 4
// 5
// string

array.forEach(function(value, index) {
  // podemos acceder al índice
  console.log(value, index);
});
// Resultado:
// 3 0
// 4 1
// 5 2
```

```
// STRING
for (value of string) {
  console.log(value);
}
// Resultado:
// s
// t
// r
// i
// n
// g

string.forEach(function(value, index) {
  console.log(value, index);
});
// Resultado:
// TypeError: string.forEach is not a function
```

Ejercicio

Sign in

GitLab.com

https://gitlab.com/upgrade-hub/ejercicios-master/javascript/javascript-sesion-3

Continuamos con la sesión 4: