



# JS | Bucles y Condicionales

**Después de esta lección podrás:**

1. Entender e identificar valores de tipo booleano.
2. Usar operadores booleanos: **and**, **or** y **not**.
3. Comprender y aplicar los condicionales en código.
4. Entender y usar **if..else**.
5. Aplicar **bucles** y entender la necesidad en código.
6. Entender y usar **while**.
7. Entender y usar **for**.

En Javascript existen diferentes tipos de valores, con los que trabajaremos. Ya vimos los números y las cadenas (o strings) de las que hemos hablado en la parte anterior, ahora nos centraremos en valores **booleanos** que son los que explicaremos a continuación.

## Boolean

El tipo booleano es bastante simple, solo tiene dos posibilidades: sí o no. Por ejemplo:

```
var Hulk = true; // Valor verdadero
var teamIronMan = false; // Valor falso
```

Los booleanos los usaremos en las declaraciones **condicionales** que veremos en esta unidad.

Pero antes, vamos a echar un vistazo a los operadores lógicos, los cuales nos permitirán enriquecer nuestras preguntas o condiciones.

## Operadores lógicos booleanos

Un operador lógico nos va a permitir **combinar** valores booleanos. Esto nos será útil para operar con ellos. En anteriores secciones ya vimos como operar con números o cadenas de texto, recordémoslo:

```
var suma = 2 + 2;
var texto = "Hola" + " " + "me llamo Juan";
```

Tenemos tres operadores lógicos diferentes: or, and y not.

### Operador OR (||)

El operador **or**, representado por `||`, devuelve verdadero si uno de los valores combinados es verdadero.

```
var tengoEfectivo = true;
var tengoTarjeta = false;
```

```
var puedoPagar = tengoEfectivo || tengoTarjeta;  
console.log(puedoPagar); // Devuelve true, porque tengo efectivo
```

A continuación os dejamos todas las posibles combinaciones con OR:

```
true  || true    // => true  
true  || false   // => true  
false || true    // => true  
false || false   // => false
```

## Operador AND (&&)

El operador **and**, representado por `&&`, devuelve verdadero solo si todos los valores combinados son verdaderos.

```
var tengoCoche = false;  
var tengoCarnetDeConducir = true;  
var puedoConducir = tengoCoche && tengoCarnetDeConducir;  
console.log(puedoConducir); // Devuelve false, porque no tengo coche
```

A continuación os dejamos todas las posibles combinaciones con AND:

```
true  && true    // => true  
true  && false   // => false  
false && true    // => false  
false && false   // => false
```

## Operador NOT (!)

Por último, pero no menos importante, tenemos el operador **not** de negación `!`. Se usa para negar el valor de una expresión (darle el valor opuesto).

```
!true    // => false
!false   // => true
```

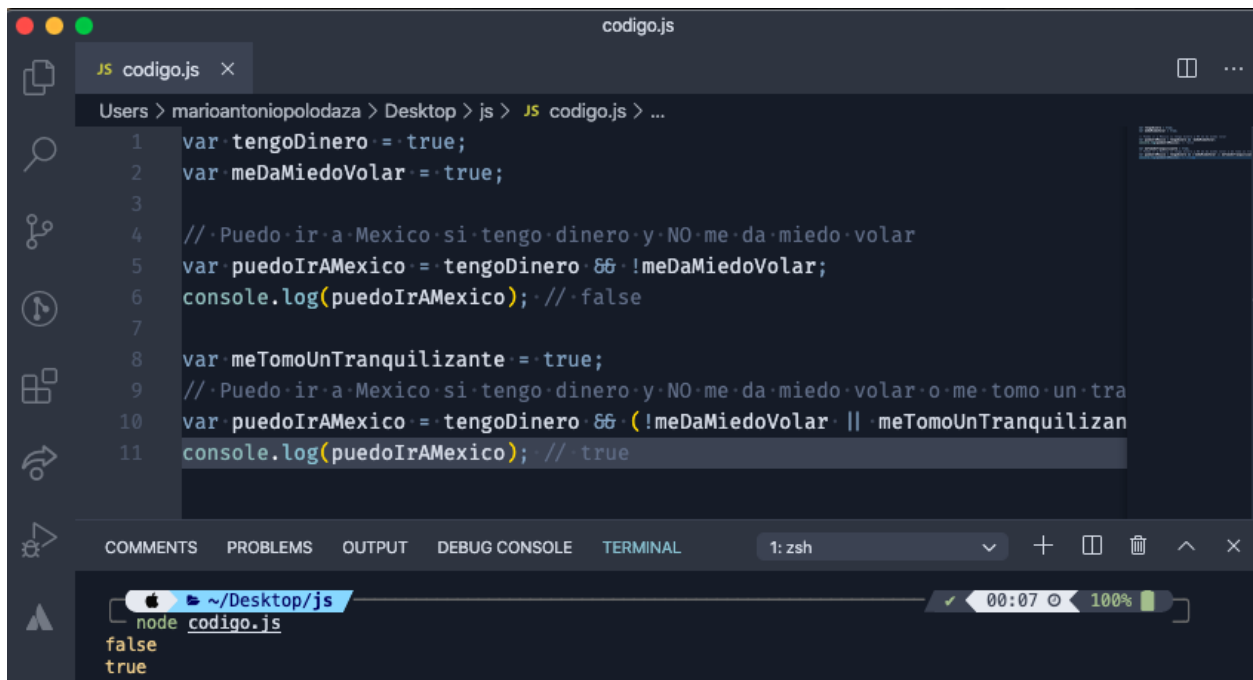
## Ejercicio con operadores lógicos

Ahora que ya hemos visto todos los operadores, vamos a realizar algunas comprobaciones, para ello debemos copiar este código en nuestro fichero de **codigo.js** y probar a cambiar los valores de las variables, para ver las posibles combinaciones.

```
var tengoDinero = true;
var meDaMiedoVolar = true;

// Puedo ir a Mexico si tengo dinero y NO me da miedo volar
var puedoIrAMexico = tengoDinero && !meDaMiedoVolar;
console.log(puedoIrAMexico);

var meTomoUnTranquilizante = true;
// Puedo ir a Mexico si tengo dinero y NO me da miedo volar o me tomo un tranquilizante
var puedoIrAMexico = tengoDinero && (!meDaMiedoVolar || meTomoUnTranquilizante);
console.log(puedoIrAMexico);
```



```
Users > marioantoniopolodaza > Desktop > js > JS codigo.js > ...  
1 var tengoDinero = true;  
2 var meDaMiedoVolar = true;  
3  
4 // Puedo ir a Mexico si tengo dinero y NO me da miedo volar  
5 var puedoIrAMexico = tengoDinero && !meDaMiedoVolar;  
6 console.log(puedoIrAMexico); // false  
7  
8 var meTomoUnTranquilizante = true;  
9 // Puedo ir a Mexico si tengo dinero y NO me da miedo volar o me tomo un tra  
10 var puedoIrAMexico = tengoDinero && (!meDaMiedoVolar || meTomoUnTranquilizan  
11 console.log(puedoIrAMexico); // true  
  
COMMENTS PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: zsh  
node ~/Desktop/js  
false  
true
```

## Operadores de comparación

Ahora que ya sabemos cómo combinar valores booleanos, vamos a ver como compararlos entre sí. Podemos comprobar si dos valores booleanos son:

- `==` iguales
- `!=` desiguales
- `===` iguales estrictamente
- `!==` desiguales estrictamente

Centrémonos en los dos primeros, los más básicos:

```
true == true // true  
true == false // false  
true != true // false  
true != false // true
```

La diferencia con los operadores más estrictos, es que hacen una comparación sin importar el **tipo** de los valores que estamos comparando:

```
1 == 1      // true
'1' == 1    // true
1 == '1'    // true
0 == false  // true

1 != 2      // true
1 != '1'    // false
1 != "1"    // false
1 != true   // false
0 != false  // false
```

Y los comparadores más estrictos analizan el **tipo** de dato, además de comparar el valor interno:

```
3 === 3    // true
3 === '3'  // false
3 !== '3'  // true
4 !== 3    // true
```

Os dejamos un [link](#) para aquel que quiera una definición más técnica, y así profundizar y entender las diferencias entre las comparaciones básicas y las estrictas.

Para finalizar el tema de comparaciones, también podremos comparar números:

- `<` menor que ...
- `>` mayor que ...
- `<=` menor o igual que ...
- `>=` mayor o igual que ...

```
2 < 4    // true
2 > 4    // false
2 <= 2   // true
2 >= 1   // true
```

Una vez hemos visto los operadores, podemos realizar **condiciones** para ejecutar una parte de nuestro código u otra. Esto se realiza a través de **condicionales**.

## Condicionales

Durante la vida de un Vengador, tenemos que tomar decisiones. ¿Traje con capa o sin capa? ¿Team Cap o Team Iron Man? Algunas de ellas son más importantes que otras, pero todas tienen algo en común: tenemos que **decidir** qué hacer.

En programación, tenemos que tomar algunas decisiones también. ¿Cómo podemos ejecutar algún código **dependiendo** de la decisión de nuestros usuarios? ¿Qué pasa si pueden elegir varias opciones? Utilizamos declaraciones condicionales para hacer este tipo de cosas.

### if - if..else

Nos va a permitir comparar **si** se cumple una condición, para tomar un camino, y **sino** tomar otro. Tan simple como eso, gracias a esta sentencia podemos dividir nuestro código en dos caminos, uno para el supuesto verdadero y otro para el falso:

```
var age = 15;
if (age === 15) {
  console.log("Mi edad es 15"); // Este mensaje se imprime por pantalla
}

if (age === 18) {
  console.log("Mi edad es 18"); // Este mensaje no se imprime, porque mi edad no es 18
}
```

```
// Edad que tenemos
var age = 15;
// Si soy mayor de edad, entonces puedo ser un Vengador
if (age < 18) {
  // Si mi edad es menor de 18
  console.log ("Vaya tendrás que ir con Spedy a jugar al parque");
} else {
  // Si mi edad es mayor de 18
  console.log ("Bienvenido Vengador");
}
```

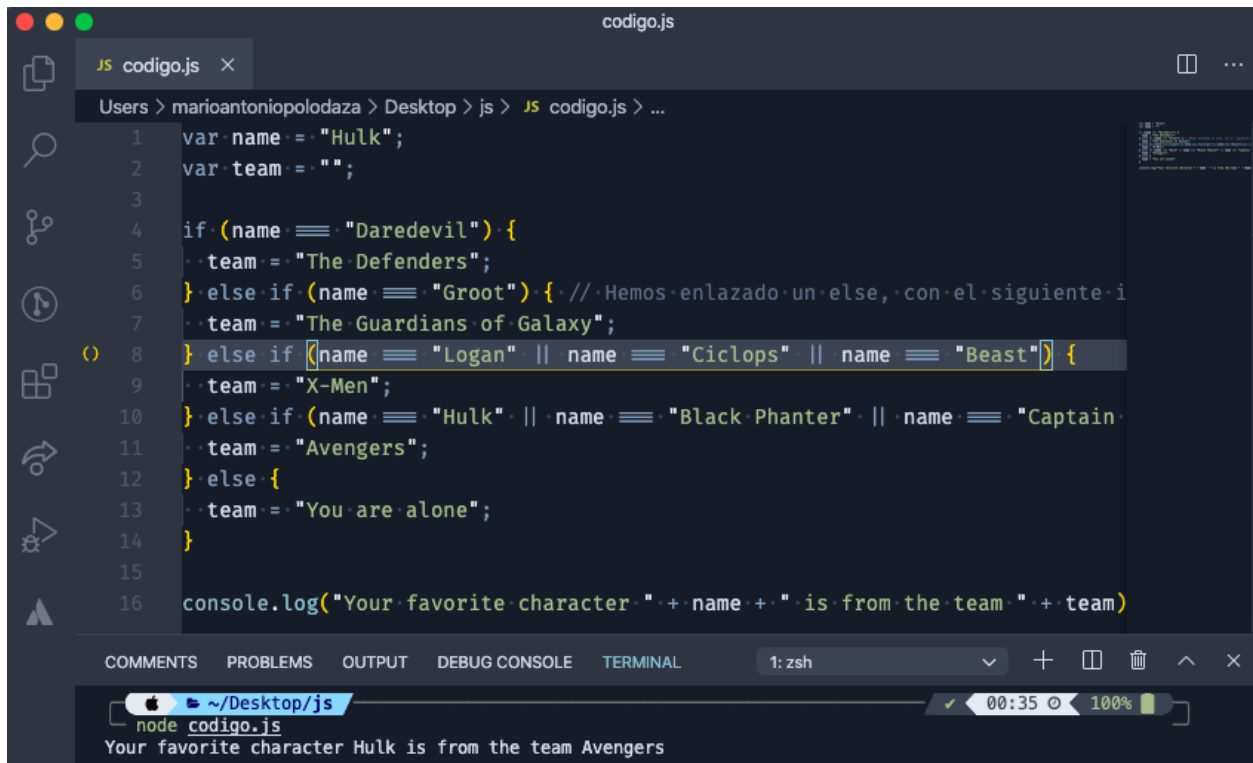
A veces, una declaración `if..else` puede volverse muy complicada. Supongamos que queremos descubrir el equipo de nuestro personaje principal favorito de Marvel. En tal caso debemos ir **enlazando** comparaciones, para abrirnos varias posibilidades:

```
var name = "Hulk";
var team = "";

if (name === "Daredevil") {
  team = "The Defenders";
} else if (name === "Groot") { // Hemos enlazado un else, con el siguiente if
  team = "The Guardians of Galaxy";
} else if (name === "Logan" || name === "Ciclops" || name === "Beast") {
  team = "X-Men";
} else if (name === "Hulk" || name === "Black Phanter" || name === "Captain America") {
  team = "Avengers";
} else {
  team = "You are alone";
}

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





```
1 var name = "Hulk";
2 var team = "";
3
4 if (name === "Daredevil") {
5   team = "The Defenders";
6 } else if (name === "Groot") { // Hemos enlazado un else, con el siguiente if
7   team = "The Guardians of Galaxy";
8 } else if (name === "Logan" || name === "Ciclops" || name === "Beast") {
9   team = "X-Men";
10 } else if (name === "Hulk" || name === "Black Panther" || name === "Captain
11   team = "Avengers";
12 } else {
13   team = "You are alone";
14 }
15
16 console.log("Your favorite character " + name + " is from the team " + team)
```

COMMENTS PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: zsh

node codigo.js  
Your favorite character Hulk is from the team Avengers

## Bucles e iteraciones

Utilizaremos bucles e iteraciones para realizar tareas repetitivas en nuestros programas. Por ejemplo, si queremos imprimir los primeros 100 números, no los escribiremos todos. Usaremos un bucle `while` o para hacer eso. Nos ofrecen una manera rápida y fácil de hacer algo repetidamente.

### while

```
var i = 0;

// Mientras la variable "i" sea menor o igual que 100
while (i <= 100) {
  console.log(i);
  i = i + 1; // Suma 1 a la variable i
}
```

El bucle sigue las siguientes reglas:

- Comprueba si el valor de `i` es menor o igual que 100.
- Imprime en la consola el valor de `i`.
- Incrementa el valor de la `i` en 1.

## for

En caso de necesitar un bucle algo más robusto y sofisticado, usaremos la instrucción `for`. Con ella crearemos un bucle con tres valores diferentes separados por punto y coma: **inicialización**, **condición** y **expresión de incremento**. Es un poco complicado al principio pero a través de un ejemplo sencillo lo verás mejor.

Vamos a hacer el mismo ejercicio que antes, esta vez usando una declaración **for**. Imprimamos en la consola los números del 0 al 100. El código es:

```
// Inicializacion: var i = 0
// Condición: si i es menor o igual que 100
// Incremento: por cada iteración, sumale 1 al valor de 'i'
// Pista extra: i++ es lo mismo que: i = i + 1
for (var i = 0; i <= 100; i++) {
  console.log(i);
}
```

## Resumen

Para resumir, en esta unidad de aprendizaje hemos aprendido que los booleanos son variables que representan dos valores diferentes: verdadero o falso.

Hemos aprendido cómo operar con esos valores y a compararlos. Todo ello para usar expresiones en las declaraciones condicionales como **if..else**.

Por último, hemos visto cómo realizar bucles de cara a ahorrarnos escribir varias

veces las mismas sentencias, con **while** y **for**.

## Recursos Extra

- [MDN - Logical operators](#)
- [MDN - `if...else` statement](#)
- [MDN - `for` statement](#)
- [MDN - `while` statement](#)

## ¡Hora de hacer ejercicios!

Vamos a practicar todo lo aprendido hasta ahora en: