

SOC

Servei d'Ocupació  
de Catalunya



Generalitat  
de Catalunya



Unió Europea  
Fons social europeu  
L'FSE inverteix en el teu futur



## MÓDULO 1. MF0951\_2 INTEGRAR COMPONENTES SOFTWARE EN PÁGINAS WEB

### UNIDAD FORMATIVA 1. UF1305 INTEGRAR COMPONENTES SOFTWARE EN PÁGINAS WEB.





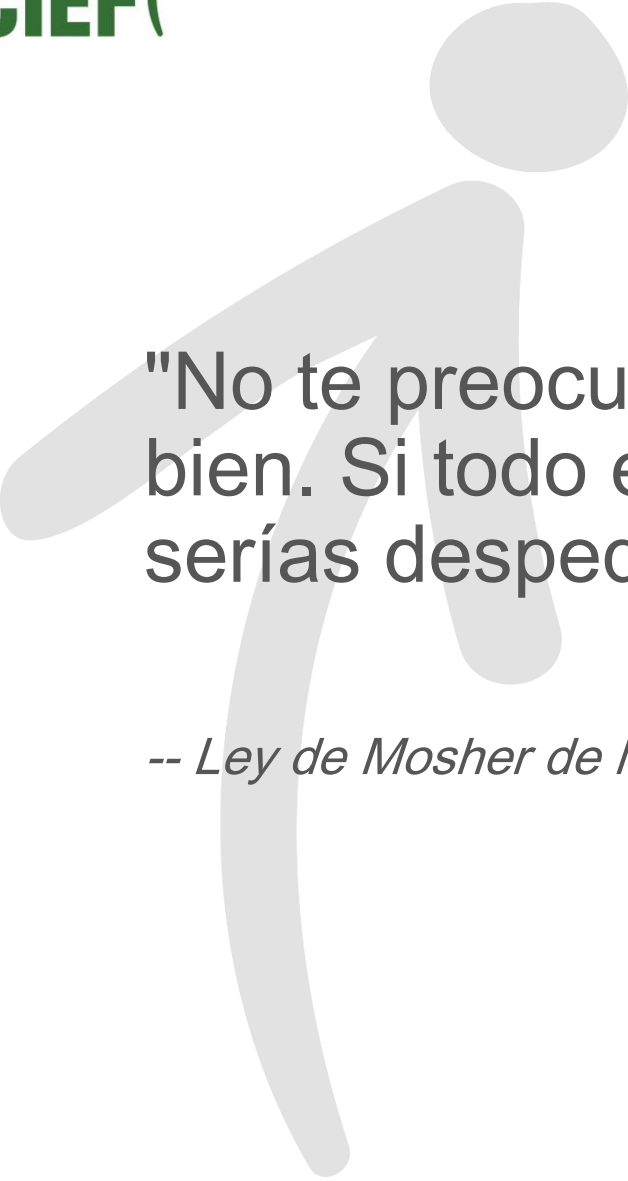


### 3.3

#### Operadores y expresiones.

- \_Salida de datos
- \_Entrada de datos
- \_Operadores de asignación.
- \_Operadores de comparación.
- \_Operadores aritméticos.
- \_Operadores lógicos.
- \_Operadores de cadenas de caracteres.
- \_Operadores especiales.
- \_Ejercicio con operadores
- \_Expresiones de cadena.
- \_Expresiones aritméticas.
- \_Expresiones lógicas.
- \_Expresiones de objeto.





"No te preocupes si no funciona bien. Si todo estuviera correcto, serías despedido de tu trabajo"

-- Ley de Mosher de la Ingeniería del Software

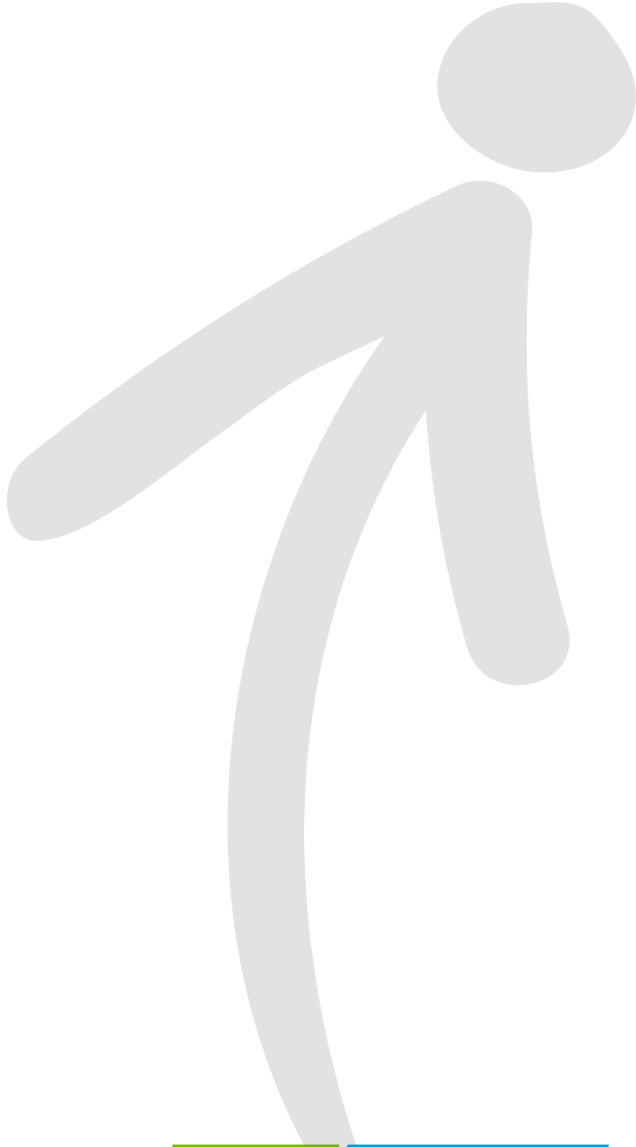


# Javascript.I



**3.3.**

**Operadores**



Index\_5



### 3.3 Operadores y expresiones\_ Operadores de asignación

Las variables por sí solas son de poca utilidad.

Hasta ahora, sólo se ha visto cómo crear variables de diferentes tipos .

**Para hacer programas realmente útiles, son necesarias otro tipo de herramientas.**

Los operadores permiten manipular el valor de las variables, realizar operaciones matemáticas con sus valores y comparar diferentes variables.

De esta forma, los operadores permiten a los programas realizar cálculos complejos y tomar decisiones lógicas en función de comparaciones y otros tipos de condiciones.

## 3.3 Operadores y expresiones\_ Operadores de asignación

Nombre	Operador abreviado	Significado
Asignación	<code>x = y</code>	<code>x = y</code>
Asignación de adición	<code>x += y</code>	<code>x = x + y</code>
Asignación de resta	<code>x -= y</code>	<code>x = x - y</code>
Asignación de multiplicación	<code>x *= y</code>	<code>x = x * y</code>
Asignación de división	<code>x /= y</code>	<code>x = x / y</code>
Asignación de residuo	<code>x %= y</code>	<code>x = x % y</code>
Asignación de exponenciación	<code>x **= y</code>	<code>x = x ** y</code>

## 3.3 Operadores y expresiones\_ Operadores de asignación

Asignación de desplazamiento a la izquierda	<code>x &lt;&lt;= y</code>	<code>x = x &lt;&lt; y</code>
Asignación de desplazamiento a la derecha	<code>x &gt;&gt;= y</code>	<code>x = x &gt;&gt; y</code>
Asignación de desplazamiento a la derecha sin signo	<code>x &gt;&gt;&gt;= y</code>	<code>x = x &gt;&gt;&gt; y</code>
Asignación AND bit a bit	<code>x &amp;= y</code>	<code>x = x &amp; y</code>
Asignación XOR bit a bit	<code>x ^= y</code>	<code>x = x ^ y</code>
Asignación OR bit a bit	<code>x  = y</code>	<code>x = x   y</code>
Asignación AND lógico	<code>x &amp;&amp;= y</code>	<code>x &amp;&amp; (x = y)</code>
Asignación OR lógico	<code>x   = y</code>	<code>x    (x = y)</code>
Asignación de anulación lógica	<code>x ??= y</code>	<code>x ?? (x = y)</code>



### Variables. Let, var y constantes

**VAR:** Es una variable que SI puede cambiar su valor y su scope es local.

**LET:** Es una variable que también podrá cambiar su valor, pero solo vivirá (Funcionará) en el bloque donde fue declarada. Por ejemplo en un bucle

**CONST:** Es una variable constante la cual NO cambiara su valor en ningún momento en el futuro.

40
50
50
texto1
texto2
texto1
>

```
'use strict'

//VARIABLES
//Let y Var

// Prueba con Var
var numero = 40;
console.log(numero); //valor 40

if(true){
    var numero = 50;
    console.log(numero); //valor 50
}

console.log(numero); //valor 50

// Prueba con Let
var texto = "texto1";
console.log(texto); //valor "texto1"

if(true){
    let texto = "texto2";
    console.log(texto); //valor "texto2"
}

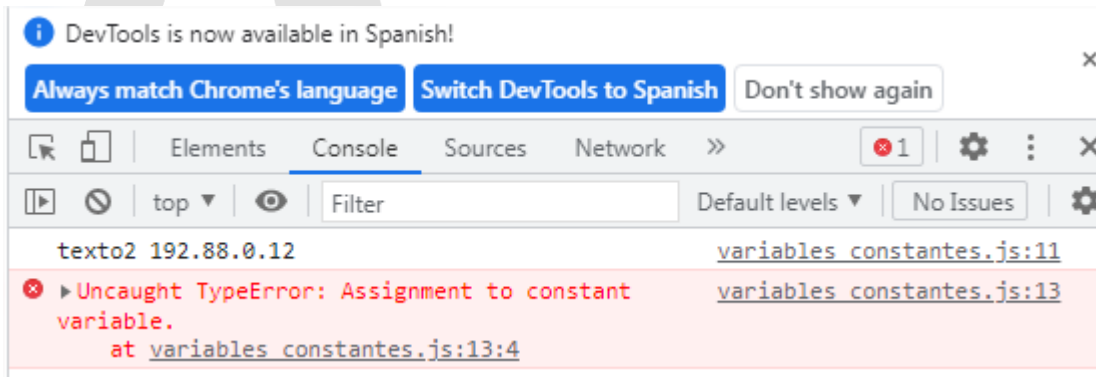
console.log(texto); //valor "texto1"
```

Index\_4



### Variables. Let, var y constantes

**CONST:** Es una variable constante la cual NO cambiara su valor en ningún momento en el futuro.



```
'use strict'
```

```
//VARIABLES  
//constantes
```

```
var web = "texto1";  
const ip = "192.88.0.12";
```

```
web = "texto2"; //ok
```

```
console.log(web, ip);
```

```
ip = "0.0.0.0"; //no lo permite da error
```

Los operadores relacionales realizan comparaciones entre sus operandos.

Se usan habitualmente dentro de expresiones condicionales.

Todos estos operadores se evalúan a verdadero o falso.

Operador	Descripción
==	" Igual a" devuelve true si los operandos son iguales
===	Estrictamente "igual a" (JavaScript 1.3)
!=	" No igual a" devuelve true si los operandos no son iguales
!==	Estrictamente " No igual a" (JavaScript 1.3)
>	" Mayor que" devuelve true si el operador de la izquierda es mayor que el de la derecha.
>=	" Mayor o igual que " devuelve true si el operador de la izquierda es mayor o igual que el de la derecha.
<	" Menor que" devuelve true si el operador de la izquierda es menor que el de la derecha.
<=	"Menor o igual que" devuelve true si el operador de la izquierda es menor o igual que el de la derecha.

Operador	Descripción	Ejemplos que devuelven <code>true</code>
Igual ( <code>==</code> )	Devuelve <code>true</code> si los operandos son iguales.	<code>3 == var1</code> <code>"3" == var1</code> <code>3 == '3'</code>
No es igual ( <code>!=</code> )	Devuelve <code>true</code> si los operandos <i>no</i> son iguales.	<code>var1 != 4</code> <code>var2 != "3"</code>
Estrictamente igual ( <code>===</code> )	Devuelve <code>true</code> si los operandos son iguales y del mismo tipo. Consulta también <a href="#">Object.is</a> y <a href="#">similitud en JS</a> .	<code>3 === var1</code>
Desigualdad estricta ( <code>!==</code> )	Devuelve <code>true</code> si los operandos son del mismo tipo pero no iguales, o son de diferente tipo.	<code>var1 !== "3"</code> <code>3 !== '3'</code>



Mayor que ( > )	Devuelve <code>true</code> si el operando izquierdo es mayor que el operando derecho.	<code>var2 &gt; var1</code> <code>"12" &gt; 2</code>
Mayor o igual que ( >= )	Devuelve <code>true</code> si el operando izquierdo es mayor o igual que el operando derecho.	<code>var2 &gt;= var1</code> <code>var1 &gt;= 3</code>
Menor que ( < )	Devuelve <code>true</code> si el operando izquierdo es menor que el operando derecho.	<code>var1 &lt; var2</code> <code>"2" &lt; 12</code>
Menor o igual ( <= )	Devuelve <code>true</code> si el operando izquierdo es menor o igual que el operando derecho.	<code>var1 &lt;= var2</code> <code>var2 &lt;= 5</code>



## 3.3 Operadores y expresiones\_ Operadores de comparación

Operaciones. Operador “===”

### El operador ‘===’

El operador ‘===’ no es habitual en los lenguajes de programación.

Se diferencia de ‘==’ en que además del valor comprueba el tipo de dato.

Es decir, las dos variables que se comparan no solo tienen que tener el mismo valor (después de conversiones), sino que deben tener el mismo tipo de dato.

Este operador existe porque algunas de las conversiones automáticas entre tipos de datos en JavaScript pueden ser poco intuitivas.

Lo mismo se puede decir del operador ‘!=’. Devuelve true si los operandos no son iguales y/o no son del mismo tipo.

Los operadores aritméticos se ocupan de realizar operaciones matemáticas.

Operador	Nombre	Ejemplo	Descripción
+	Suma	5 + 6	Suma dos números
-	Substracción	7 - 9	Resta dos números
*	Multiplicación	6 * 3	Multiplica dos números
/	División	4 / 8	Divide dos números
%	Módulo: el resto después de la división	7 % 2	Devuelve el resto de dividir ambos números, en este ejemplo el resultado es 1
++	Incremento.	a++	Suma 1 al contenido de una variable.
--	Decremento.	a--	Resta 1 al contenido de una variable.
-	Invierte el signo de un operando.	-a	Invierte el signo de un operando.

## Operaciones. Operadores de asignación y cálculo

Ahora que conocemos los operadores aritméticos podemos ver otros operadores asignación que **combinan la asignación con otra operación aritmética** como método abreviado.

Operador	Ejemplo	Expresión equivalente
$+=$	$x += 5$	$x = x + 5$
$-=$	$x -= 5$	$x = x - 5$
$*=$	$x *= 5$	$x = x * 5$
$/=$	$x /= 5$	$x = x / 5$
$\% =$	$x \% = 5$	$x = x \% 5$

## Operaciones. Precedencia de operadores

Precedencia	Operadores	Asociatividad
0	Operador de agrupamiento, [...]	No tiene
3	Incremento [++] y decremento [--]	No tienen
4	Negación [!]	Derecha a izquierda
5	Multiplicación [*], división [/], módulo [%]	Izquierda a derecha
6	Suma [+], resta [-]	Izquierda a derecha
8	Relacionales [ >, <, >=, <= ]	Izquierda a derecha
9	Igualdad [==, !=, ===, !==]	Izquierda a derecha
13	Y lógico [&&]	Izquierda a derecha
14	O lógico [  ]	Izquierda a derecha
17	Asignación [=]	Derecha a izquierda

## Operaciones. Operadores de incremento y decremento

Los operadores '++' y '--' suman y restan uno a la variable a la que se aplican. En concreto, las siguientes líneas

++	Incremento.	a++	Suma 1 al contenido de una variable.
--	Decremento.	a--	Resta 1 al contenido de una variable.
-	Invierte el signo de un operando.	-a	Invierte el signo de un operando.

El comportamiento de los operadores varía según se sitúen a la derecha o a la izquierda de la variable. Lo mejores verlo con un ejemplo:

```
var num1 = 0, num2= 0;  
num1 = num1 + 1;  
num2= num2 - 1;
```

son equivalentes

```
var num1 = 0, num2= 0;  
num1++;  
num2--;
```



Operaciones. Precedencia de operadores

$$a = 2 + 3 * 4$$

14

$$a = (2 + 3) * 4$$

20

$$a = ((2 + 3) * 4) + 1) * 5$$

105

4.3. Operadores\_Ope



### 3.3 Operadores y expresiones\_ Operadores de aritméticos

Además de las operaciones aritméticas estándar (+, -, \*, /), JavaScript proporciona los operadores aritméticos enumerados en la siguiente tabla:

Operador	Descripción	Ejemplo
Residuo (%)	Operador binario. Devuelve el resto entero de dividir los dos operandos.	12 % 5 devuelve 2.
Incremento (++)	Operador unario. Agrega uno a su operando. Si se usa como operador prefijo (++x), devuelve el valor de su operando después de agregar uno; si se usa como operador sufijo (x++), devuelve el valor de su operando antes de agregar uno.	Si x es 3, ++x establece x en 4 y devuelve 4, mientras que x++ devuelve 3 y, solo entonces, establece x en 4.
Decremento (--)	Operador unario. Resta uno de su operando. El valor de retorno es análogo al del operador de incremento.	Si x es 3, entonces --x establece x en 2 y devuelve 2, mientras que x-- devuelve 3 y, solo entonces, establece

NOT a nivel de bits	$\sim a$	Invierte los bits de su operando.
Desplazamiento a la izquierda	$a \ll b$	Desplaza $a$ en representación binaria $b$ bits hacia la izquierda, desplazándose en ceros desde la derecha.
Desplazamiento a la derecha de propagación de signo	$a \gg b$	Desplaza $a$ en representación binaria $b$ bits a la derecha, descartando los bits desplazados.
Desplazamiento a la derecha de relleno cero	$a \ggg b$	Desplaza $a$ en representación binaria $b$ bits hacia la derecha, descartando los bits desplazados y desplazándose en ceros desde la izquierda.



<b>Negación unaria</b> <code>( - )</code>	Operador unario. Devuelve la negación de su operando.	Si <code>x</code> es 3, entonces <code>-x</code> devuelve -3.
<b>Positivo unario</b> <code>( + )</code>	Operador unario. Intenta convertir el operando en un número, si aún no lo es.	<code>+"3"</code> devuelve 3. <code>+true</code> devuelve 1.
<b>Operador de exponenciación</b> <code>( ** )</code>	Calcula la <code>base</code> a la potencia de <code>exponente</code> , es decir, <code>base<sup>exponente</sup></code>	<code>2 ** 3</code> returns 8. <code>10 ** -1</code> returns 0.1.

Los operadores lógicos actúan sobre valores lógicos y sirven para escribir expresiones complejas.

Operador	Descripción	Ejemplo de uso
&&	Operador Y (AND). Verdadero si los dos operandos son verdaderos, falso en otro caso	var1 && var2
	Operador O (OR). Verdadero si al menos uno de los dos operando es verdaderos, falso en otro caso	var1    var2
!	Negación (NOT). Niega la variable. Si es verdadera, se evalúa a falso y viceversa	!var1
&	AND a nivel de bit	var1 & var2
	OR a nivel de bit	var1   var2
~	Negación a nivel de bit	~var1
^	O exclusivo (XOR) a nivel de bit	var1 ^ var2

Los operadores lógicos se utilizan normalmente con valores booleanos (lógicos); cuando lo son, devuelven un valor booleano.

Sin embargo, los operadores `&&` y `||` en realidad devuelven el valor de uno de los operandos especificados, por lo que si estos operadores se utilizan con valores no booleanos, pueden devolver un valor no booleano

Operadores lógicos

Operador	Uso	Descripción
AND lógico ( && )	<code>expr1 &amp;&amp; expr2</code>	Devuelve <code>expr1</code> si se puede convertir a <code>false</code> ; de lo contrario, devuelve <code>expr2</code> . Por lo tanto, cuando se usa con valores booleanos, <code>&amp;&amp;</code> devuelve <code>true</code> si ambos operandos son <code>true</code> ; de lo contrario, devuelve <code>false</code> .
OR lógico (    )	<code>expr1    expr2</code>	Devuelve <code>expr1</code> si se puede convertir a <code>true</code> ; de lo contrario, devuelve <code>expr2</code> . Por lo tanto, cuando se usa con valores booleanos, <code>  </code> devuelve <code>true</code> si alguno de los operandos es <code>true</code> ; si ambos son falsos, devuelve <code>false</code> .
NOT lógico ( ! )	<code>!expr</code>	Devuelve <code>false</code> si su único operando se puede convertir a <code>true</code> ; de lo contrario, devuelve <code>true</code> .

## 3.3 Operadores y expresiones\_ Operadores lógicos

El siguiente código muestra ejemplos del operador && (AND lógico).

```
var a1 = true && true;    // t && t devuelve true
var a2 = true && false;   // t && f devuelve false
var a3 = false && true;    // f && t devuelve false
var a4 = false && (3 == 4); // f && f devuelve false
var a5 = 'Cat' && 'Dog';   // t && t devuelve Dog
var a6 = false && 'Cat';   // f && t devuelve false
var a7 = 'Cat' && false;   // t && f devuelve false
```

El siguiente código muestra ejemplos del operador || (OR lógico).

```
var o1 = true || true;    // t || t devuelve true
var o2 = false || true;   // f || t devuelve true
var o3 = true || false;   // t || f devuelve true
var o4 = false || (3 == 4); // f || f devuelve false
var o5 = 'Cat' || 'Dog';   // t || t devuelve Cat
var o6 = false || 'Cat';   // f || t devuelve Cat
var o7 = 'Cat' || false;   // t || f devuelve Cat
```

## 3.3 Operadores y expresiones\_ Operadores lógicos

El siguiente código muestra ejemplos de el operador ! (NOT lógico).

```
var n1 = !true; // !t devuelve false  
var n2 = !false; // !f devuelve true  
var n3 = !'Cat'; // !t devuelve false
```

## 3.3 Operadores y expresiones\_ Operadores de cadenas

Además de los operadores de comparación, que se pueden usar en valores de cadena, el operador de concatenación (+) concatena dos valores de cadena, devolviendo otra cadena que es la unión de los dos operandos de cadena.

Por ejemplo,

```
console.log('mi ' + 'cadena'); // la consola registra la cadena "mi cadena".
```

El operador de asignación abreviada += también se puede utilizar para concatenar cadenas.

Por ejemplo,

```
var mystring = 'alpha';  
mystring += 'bet'; // se evalúa como "alphabet" y asigna este valor a mystring.
```

## 3.3 Operadores y expresiones\_ Operadores especiales

### Operador condicional (ternario)

El operador condicional es el único operador de JavaScript que toma tres operandos. El operador puede tener uno de dos valores según una condición. La sintaxis es:

```
condition ? val1 : val2
```

Si *condition* es true, el operador tiene el valor de *val1*. De lo contrario, tiene el valor de *val2*. Puedes utilizar el operador condicional en cualquier lugar donde normalmente utilizas un operador estándar.

Por ejemplo,

```
var status = (age >= 18) ? 'adult' : 'minor';
```

## 3.3 Operadores y expresiones\_ Operadores especiales

### Operador Nullish Coalescing '??'

El resultado de `a ?? b`:

- si `a` está “definida”, será `a`,
- si `a` no está “definida”, será `b`.

Es decir, `??` devuelve el primer argumento cuando este no es `null` ni `undefined`. En caso contrario, devuelve el segundo.

El operador “nullish coalescing” no es algo completamente nuevo. Es solamente una sintaxis agradable para obtener el primer valor “definido” de entre dos.

Podemos reescribir `result = a ?? b` usando los operadores que ya conocemos:

```
result = (a !== null && a !== undefined) ? a : b;
```

```
let user;  
alert(user ?? "Anonymous"); // Anonymous (user no definido)
```

```
let user = "John";  
alert(user ?? "Anonymous"); // John (user definido)
```



## 3.3 Operadores y expresiones\_ Operadores especiales

### Operador Nullish Coalescing '??'

Digamos que tenemos los datos de un usuario en las variables `firstName`, `lastName` y `nickName`. Todos ellos podrían ser indefinidos si el usuario decide no ingresarlos.

Queremos mostrar un nombre usando una de las tres variables, o mostrar “anónimo” si ninguna está definida: Usemos el operador `??` para ello:

```
let user;  
let firstName = null;  
let lastName = null;  
let nickName = "Supercoder";  
// Muestra el primer valor definido:  
alert(firstName ?? lastName ?? nickName ?? "Anonymous"); // Supercoder
```

```
'use strict'
```

```
//OPERADORES
```

```
var num1 = 7;
```

```
var num2 = 12;
```

```
var operacion = num1 * num2;
```

```
alert("Resultado :“ + operacion);
```

```
//Tipo de datos
```

```
var numero_entero = 44;
```

```
var numero_decimal = 55.4;
```

```
var cadena_texto = 'Hola "que" tal';
```

```
var aceptado = false;
```

```
console.log(aceptado);
```

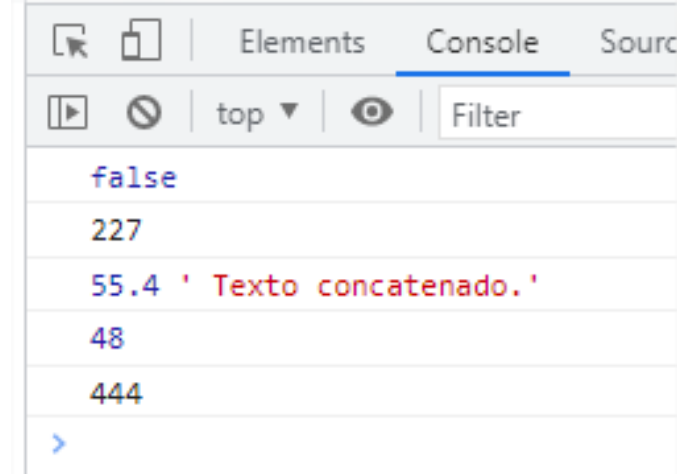
```
var numero_falso = "22";
```

```
console.log(numero_falso+7);
```

```
console.log(numero_decimal,' Texto concatenado.');
```

```
console.log(numero_entero+4);
```

```
console.log(String(numero_entero)+4);
```



3.3.

# Expresiones



### Longitud de cadena de JavaScript

Para encontrar la longitud de una cadena, use la **length** propiedad incorporada:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Properties</h2>

<p>The length property returns the length of a string:</p>

<p id="demo"></p>

<script>
let text = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
document.getElementById("demo").innerHTML = text.length;
</script>

</body>
</html>
```

### JavaScript String Properties

The **length** property returns the length of a string:

26



### Segmento de cadena de JavaScript ()

**slice()** extrae una parte de una cadena y devuelve la parte extraída en una nueva cadena.

El método toma 2 parámetros: la posición inicial y la posición final (final no incluido). Este ejemplo corta una porción de una cadena desde la posición 7 a la posición 12 (13-1):

```
let str = "Apple, Banana, Kiwi";  
let part = str.slice(7, 13);
```

```
<!DOCTYPE html>  
<html>  
<body>  
  
<h2>JavaScript String Methods</h2>  
  
<p>The slice() method extract a part of a string  
and returns the extracted parts in a new string:</p>  
  
<p id="demo"></p>  
  
<script>  
let str = "Apple, Banana, Kiwi";  
document.getElementById("demo").innerHTML = str.slice(7,13);  
</script>  
  
</body>  
</html>
```

### JavaScript String Methods

The slice() method extract a part of a string and returns the extracted parts in a new string:

Banana



### Subcadena de cadena de JavaScript ()

**substring()** es similar a slice().

La diferencia es que substring() no puede aceptar índices negativos.

```
let str = "Apple, Banana, Kiwi";  
let part = str.substring(7, 13);
```

```
<!DOCTYPE html>  
<html>  
<body>  
  
<h2>JavaScript String Methods</h2>  
  
<p>The substring() method extract a part of a string and returns the extracted parts in a  
new string:</p>  
  
<p id="demo"></p>  
  
<script>  
let str = "Apple, Banana, Kiwi";  
document.getElementById("demo").innerHTML = str.substring(7,13);  
</script>  
  
</body>  
</html>
```

### JavaScript String Methods

The slice() method extract a part of a string and returns the extracted parts in a new string:

Banana



## Substr de cadena de JavaScript ()

**substr()** es similar a slice().

La diferencia es que el segundo parámetro especifica la **longitud** de la parte extraída.

```
let str = "Apple, Banana, Kiwi";  
let part = str.substr(7, 6);
```

```
<!DOCTYPE html>  
<html>  
<body>  
  
<h2>JavaScript String Methods</h2>  
  
<p>The substr() method extract a part of a string  
and returns the extracted parts in a new string:</p>  
  
<p id="demo"></p>  
  
<script>  
let str = "Apple, Banana, Kiwi";  
document.getElementById("demo").innerHTML = str.substr(7,6);  
</script>  
  
</body>  
</html>
```

### JavaScript String Methods

The slice() method extract a part of a string and returns the extracted parts in a new string:

Banana

Si omite el segundo parámetro, **substr()** cortará el resto de la cadena.



## Sustitución del contenido de la cadena

El **replace()** método reemplaza un valor especificado con otro valor en una cadena:

```
<!DOCTYPE html>
<html>

<body>

<h2>JavaScript String Methods</h2>

<p>Replace "Microsoft" with "HP" in the paragraph below:</p>

<button onclick="myFunction()">Try it</button>

<p id="demo">Please visit Microsoft!</p>

<script>
function myFunction() {
  let text = document.getElementById("demo").innerHTML;
  document.getElementById("demo").innerHTML =
    text.replace("Microsoft","HP");
}
</script>

</body>
</html>
```

```
let text = "Please visit Microsoft!";
let newText = text.replace("Microsoft", "HP");
```

### JavaScript String Methods

Replace "Microsoft" with "HP" in the paragraph below:

Try it

Please visit Microsoft!

### JavaScript String Methods

Replace "Microsoft" with "HP" in the paragraph below:

Try it

Please visit HP!

El **replace()** método no cambia la cadena a la que se llama.

El **replace()** método devuelve una nueva cadena.

El **replace()** método reemplaza **solo la primera** coincidencia.





### Conversión a mayúsculas y minúsculas

Una cadena se convierte a mayúsculas con toUpperCase():

Una cadena se convierte a minúsculas con toLowerCase():

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Methods</h2>
<p>Convert string to upper case:</p>

<button onclick="myFunction()">Try it</button>

<p id="demo">Hello World!</p>

<script>
function myFunction() {
  let text = document.getElementById("demo").innerHTML;
  document.getElementById("demo").innerHTML =
    text.toUpperCase();
}
</script>

</body>
</html>
```

```
let text1 = "Hello World!";
let text2 = text1.toUpperCase();
```

```
let text1 = "Hello World!";    // String
let text2 = text1.toLowerCase(); // text2 is text1
                                converted to lower
```



JavaScript Cadena concatenación ()  
concat() une dos o más cadenas:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Methods</h2>

<p>The concat() method joins two or more strings:</p>

<p id="demo"></p>

<script>
let text1 = "Hello";
let text2 = "World!";
let text3 = text1.concat(" ",text2);
document.getElementById("demo").innerHTML = text3;
</script>

</body>
</html>
```

```
let text1 = "Hello";
let text2 = "World";
let text3 = text1.concat(" ", text2);
```

### JavaScript String Methods

The `concat()` method joins two or more strings:

Hello World!

El `concat()` método se puede utilizar en lugar del operador más. Estas dos líneas hacen lo mismo:

```
text = "Hello" + " " + "World!";
text = "Hello".concat(" ", "World!");
```



### Recorte de cadena de JavaScript ()

El **trim()** método elimina los espacios en blanco de ambos lados de una cadena:

```
<!DOCTYPE html>
<html>
<body>

<h1>JavaScript Strings</h1>
<h2>The trim() Method</h2>

<p id="demo"></p>

<script>
let text1 = "  Hello World!  ";
let text2 = text1.trim();

document.getElementById("demo").innerHTML =
"Length text1=" + text1.length + "<br>Length2 text2=" + text2.length;
</script>

</body>
</html>
```

```
let text1 = "  Hello World!  ";
let text2 = text1.trim();
```

### JavaScript Strings

#### The trim() Method

```
Length text1=22
Length2 text2=12
```



### Relleno de cadena de JavaScript

**padStart** y **padEnd** para admitir el relleno al principio y al final de una cade

```
let text = "5";  
let padded = text.padStart(4,0);
```

```
<!DOCTYPE html>  
<html>  
<body>  
  
<h2>JavaScript String Methods</h2>  
  
<p>The padStart() method pads a string with another string:</p>  
  
<p id="demo"></p>  
  
<script>  
let text = "5";  
document.getElementById("demo").innerHTML = text.padStart(4,0);  
</script>  
  
</body>  
</html>
```

### JavaScript String Methods

The padStart() method pads a string with another string:

0005



### Extracción de caracteres de cadena

El **charAt()** método devuelve el carácter en un índice especificado (posición) en una cadena:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Methods</h2>

<p>The charAt() method returns the character at a given position in a string:</p>

<p id="demo"></p>

<script>
var text = "HELLO WORLD";
document.getElementById("demo").innerHTML = text.charAt(0);
</script>

</body>
</html>
```

```
let text = "HELLO WORLD";
let char = text.charAt(0);
```

### JavaScript String Methods

The **charAt()** method returns the character at a given position in a string:

H



### Extracción de caracteres de cadena

El **charCodeAt()** método devuelve el Unicode del carácter en un índice especificado en una cadena:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Methods</h2>

<p>The charCodeAt() method returns the unicode of the character at a given position in
a string:</p>

<p id="demo"></p>

<script>
let text = "HELLO WORLD";
document.getElementById("demo").innerHTML = text.charCodeAt(0);
</script>

</body>
</html>
```

```
let text = "HELLO WORLD";
let char = text.charCodeAt(0);
```

#### JavaScript String Methods

The charCodeAt() method returns the unicode of the character at a given position in a string:

72



acsii



### Propiedad de acceso

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Methods</h2>

<p>ECMAScript 5 allows property access on strings:</p>

<p id="demo"></p>

<script>
var str = "HELLO WORLD";
document.getElementById("demo").innerHTML = str[0];
</script>
</body>
</html>
```

```
let text = "HELLO WORLD";
let char = text[0];
```

### JavaScript String Methods

ECMAScript 5 allows property access on strings:

H

El acceso a la propiedad puede ser un poco **impredecible**:  
Hace que las cadenas parezcan matrices (pero no lo son)  
Si no se encuentra ningún carácter, [ ] devuelve indefinido, mientras  
que charAt() devuelve una cadena vacía.  
Es de solo lectura. str[0] = "A" no da ningún error (¡pero no funciona!)



### División de cadena de JavaScript ()

Una cadena se puede convertir en una matriz con el **split()** método:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Methods</h2>
<p>Display the first array element, after a string split:</p>

<p id="demo"></p>

<script>
let text = "a,b,c,d,e,f";
const myArray = text.split(",");
document.getElementById("demo").innerHTML = myArray[0];
</script>

</body>
</html>
```

```
text.split(",") // Split on commas
text.split(" ") // Split on spaces
text.split("|") // Split on pipe
```

### JavaScript String Methods

Display the first array element, after a string split:

a





### Métodos de búsqueda de JavaScript

#### Cadena JavaScript indexOf()

El **indexOf()** método devuelve el índice de (la posición de) la firstaparición de un texto específico en una cadena:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Methods</h2>

<p>The indexOf() method returns the position of the first occurrence of a specified
text:</p>

<p id="demo"></p>

<script>
let str = "Please locate where 'locate' occurs!";
document.getElementById("demo").innerHTML = str.indexOf("locate");
</script>

</body>
</html>
```

```
let str = "Please locate where 'locate' occurs!";
str.indexOf("locate");
```

#### JavaScript String Methods

The `indexOf()` method returns the position of the first occurrence of a specified text:

7

## Métodos de búsqueda de JavaScript

## Cadena JavaScript lastIndexOf()

El **lastIndexOf()** método devuelve el índice de la última aparición de un texto específico en una cadena:

Ambos **indexOf()** y **lastIndexOf()** devuelven -1 si no se encuentra el texto:

```
let str = "Please locate where 'locate' occurs!";  
str.lastIndexOf("John");
```

Los **lastIndexOf()** métodos buscan hacia atrás (desde el final hasta el principio), lo que significa: si el segundo parámetro es **15**, la búsqueda comienza en la posición 15 y busca hasta el principio de la cadena.

```
let str = "Please locate where 'locate' occurs!";  
str.lastIndexOf("locate", 15);
```

```
let str = "Please locate where 'locate' occurs!";  
str.lastIndexOf("locate");
```

Ambos métodos aceptan un segundo parámetro como posición inicial de la búsqueda:

```
let str = "Please locate where 'locate' occurs!";  
str.indexOf("locate", 15);
```



## Métodos de búsqueda de JavaScript

El **search()** método busca una cadena para un valor específico y devuelve la posición de la coincidencia:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Methods</h2>

<p>The search() method returns the position of the first occurrence of a specified text in
a string:</p>

<p id="demo"></p>

<script>
let str = "Please locate where 'locate' occurs!";
document.getElementById("demo").innerHTML = str.search("locate");
</script>

</body>
</html>
```

```
let str = "Please locate where 'locate' occurs!";
str.search("locate");
```

### JavaScript String Methods

The `indexOf()` method returns the position of the first occurrence of a specified text:

7

¿Te diste cuenta?



- ¿ Los dos métodos, `indexOf()` y `search()`, son **iguales**?
- ¿Aceptan los mismos argumentos (parámetros) y devuelven el mismo valor?
- Los dos métodos **NO** son iguales. Estas son las diferencias:
- El `search()` método no puede aceptar un segundo argumento de posición inicial.
  - El `indexOf()` método no puede tomar valores de búsqueda poderosos (expresiones regulares).





### Coincidencia de cadena de JavaScript ()

El método **match()** busca en una cadena una coincidencia con una expresión regular y devuelve las coincidencias, como un objeto Array.

### Métodos de búsqueda de JavaScript

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Search</h2>

<p>Search a string for "ain":</p>

<p id="demo"></p>

<script>
let text = "The rain in SPAIN stays mainly in the plain";
document.getElementById("demo").innerHTML = text.match(/ain/g);
</script>

</body>
</html>
```

```
let text = "The rain in SPAIN stays mainly in the plain";
text.match(/ain/g);
```

#### JavaScript String Search

Search a string for "ain":

ain,ain,ain



RegExp



### Incluye la cadena de JavaScript ()

### Métodos de búsqueda de JavaScript

El **includes()** método devuelve verdadero si una cadena contiene un valor especificado.

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript String Search</h2>

<p>Check if a string includes "world":</p>

<p id="demo"></p>

<p>The includes() method is not supported in Internet Explorer.</p>

<script>
let text = "Hello world, welcome to the universe.";
document.getElementById("demo").innerHTML = text.includes("world");
</script>

</body>
</html>
```

```
let text = "Hello world, welcome to the universe.";
text.includes("world");
```

### JavaScript String Search

Check if a string **includes** "world":

true

The **includes()** method is not supported in Internet Explorer.

Compruebe si una cadena incluye "mundo", comenzando la búsqueda en la posición 12:

```
let text = "Hello world, welcome to the universe.";
text.includes("world", 12);
```



La cadena de JavaScript comienza con ()

### Métodos de búsqueda de JavaScript

El **startsWith()** método regresa true si una cadena comienza con un valor específico, de lo contrario false:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Strings</h2>

<p>Check if a string starts with "Hello":</p>

<p id="demo"></p>

<p>The startsWith() method is not supported in Internet Explorer.</p>

<script>
let text = "Hello world, welcome to the universe.";
document.getElementById("demo").innerHTML = text.startsWith("Hello");
</script>

</body>
</html>
```

```
let text = "Hello world, welcome to the universe.";
text.startsWith("Hello");
```

#### JavaScript Strings

Check if a string starts with "Hello":

true

The startsWith() method is not supported in Internet Explorer.

```
let text = "Hello world, welcome to the universe.";
text.startsWith("world") // Returns false
```

```
let text = "Hello world, welcome to the universe.";
text.startsWith("world", 5) // Returns false
```

```
let text = "Hello world, welcome to the universe.";
text.startsWith("world", 6) // Returns true
```



La cadena de JavaScript comienza con ()

### Métodos de búsqueda de JavaScript

El **endsWith()** método regresa true si una cadena termina con un valor específico, de lo contrario false:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Strings</h2>

<p>Check if a string ends with "Doe":</p>

<p id="demo"></p>

<p>The endsWith() method is not supported in Internet Explorer.</p>

<script>
let text = "John Doe";
document.getElementById("demo").innerHTML = text.endsWith("Doe");
</script>

</body>
</html>
```

```
let text = "John Doe";
text.endsWith("Doe");
```

### JavaScript Strings

Check if a string ends with "Doe":

true

The endsWith() method is not supported in Internet Explorer.

### Sintaxis de Back-Tics

Los literales de plantilla usan comillas invertidas (``) en lugar de comillas ("" ) para definir una cadena:

Con los literales de plantilla , puede usar comillas simples y dobles dentro de una cadena:

Los literales de plantilla permiten cadenas de varias líneas:

Los literales de plantilla proporcionan una manera fácil de interpolar variables y expresiones en cadenas. \${...}

### Plantillas cadena de texto

```
let text = `Hello World!`;
```

```
let text = `He's often called "Johnny"`;
```

```
let text =  
`The quick  
brown fox  
jumps over  
the lazy dog`;
```





### Sustituciones de variables

Los literales de plantilla permiten variables en cadenas:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Template Literals</h2>

<p>Template literals allows variables in strings:</p>

<p id="demo"></p>

<p>Template literals are not supported in Internet Explorer.</p>

<script>
let firstName = "John";
let lastName = "Doe";

let text = `Welcome ${firstName}, ${lastName}!`;

document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
>
```

### Plantillas cadena de texto

```
let firstName = "John";
let lastName = "Doe";

let text = `Welcome ${firstName}, ${lastName}!`;
```

### JavaScript Template Literals

Template literals allows variables in strings:

Welcome John, Doe!

Template literals are not supported in Internet Explorer.



Sustituciones de variables

Plantillas cadena de texto

```
<!DOCTYPE html>
<html>
  <body>
    <script>
      let num1=0;
      let num2=0;
      num1 = num1 + 1;
      num2 = num2 + 5;
      alert (`El primer numero ahora es ${num1} `);
      alert (`El segundo numero ahora es ${num2} ` );
    </script>
  </body>
</html>
```



Sustituciones de variables

Plantillas cadena de texto

```
<!DOCTYPE html>
<html>
  <body>
    <script>
      let num1=0;
      let num2=0;
      num1 = num1 + 1;
      num2 = num2 + 5;
      alert (`El primer numero ahora es ${num1} `);
      alert (`El segundo numero ahora es ${num2} ` );
    </script>
  </body>
</html>
```



### Sustitución de expresión

Los literales de plantilla permiten expresiones en cadenas:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Template Literals</h2>

<p>Template literals allows variables in strings:</p>

<p id="demo"></p>

<p>Template literals are not supported in Internet Explorer.</p>

<script>
let price = 10;
let VAT = 0.25;
let total = `Total: ${price * (1 + VAT).toFixed(2)}`;

document.getElementById("demo").innerHTML = total;
</script>

</body>
</html>
```

### Plantillas cadena de texto

```
let price = 10;
let VAT = 0.25;

let total = `Total: ${price * (1 + VAT).toFixed(2)}`;
```

#### JavaScript Template Literals

Template literals allows variables in strings:

Total: 12.50

Template literals are not supported in Internet Explorer.



```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Template Literals</h2>

<p>Template literals allows variables in strings:</p>

<p id="demo"></p>

<p>Template literals are not supported in Internet Explorer.</p>

<script>
let header = "Templates Literals";
let tags = ["template literals", "javascript", "es6"];

let html = `<h2>${header}</h2><ul>`;

for (const x of tags) {
  html += `<li>${x}</li>`;
}

html += `</ul>`;
document.getElementById("demo").innerHTML = html;
</script>

</body>
</html>
```

### Plantillas HTML

```
let header = "Templates Literals";
let tags = ["template literals", "javascript", "es6"];

let html = `<h2>${header}</h2><ul>`;
for (const x of tags) {
  html += `<li>${x}</li>`;
}

html += `</ul>`;
```

### JavaScript Template Literals

Template literals allows variables in strings:

### Templates Literals

- template literals
- javascript
- es6

Template literals are not supported in Internet Explorer.



El **toString()** método devuelve un número como una cadena.  
Todos los métodos numéricos se pueden usar en cualquier tipo de números (literales, variables o expresiones):

```
let x = 123;  
x.toString();  
(123).toString();  
(100 + 23).toString();
```

```
<!DOCTYPE html>  
<html>  
<body>  
  
<h2>JavaScript Number Methods</h2>  
  
<p>The toString() method converts a number to a string.</p>  
  
<p id="demo"></p>  
  
<script>  
let x = 123;  
document.getElementById("demo").innerHTML =  
  x.toString() + "<br>" +  
  (123).toString() + "<br>" +  
  (100 + 23).toString();  
</script>  
  
</body>  
</html>
```

### JavaScript Number Methods

The `toString()` method converts a number to a string.

123  
123  
123



**toFixed()** devuelve una cadena, con el número escrito con un número específico de decimales:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Number Methods</h2>

<p>The toFixed() method rounds a number to a given number of digits.</p>
<p>For working with money, toFixed(2) is perfect.</p>

<p id="demo"></p>

<script>
let x = 9.656;
document.getElementById("demo").innerHTML =
  x.toFixed(0) + "<br>" +
  x.toFixed(2) + "<br>" +
  x.toFixed(4) + "<br>" +
  x.toFixed(6);
</script>

</body>
</html>
```

```
let x = 9.656;
x.toFixed(0);
x.toFixed(2);
x.toFixed(4);
x.toFixed(6);
```

### JavaScript Number Methods

The `toFixed()` method rounds a number to a given number of digits.

For working with money, `toFixed(2)` is perfect.

```
10
9.66
9.6560
9.656000
```



**toPrecision()** devuelve una cadena, con un número escrito con una longitud especificada:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Number Methods</h2>

<p>The toPrecision() method returns a string, with a number written with a specified length:</p>

<p id="demo"></p>

<script>
let x = 9.656;
document.getElementById("demo").innerHTML =
  x.toPrecision() + "<br>" +
  x.toPrecision(2) + "<br>" +
  x.toPrecision(4) + "<br>" +
  x.toPrecision(6);
</script>

</body>
</html>
```

```
let x = 9.656;
x.toPrecision();
x.toPrecision(2);
x.toPrecision(4);
x.toPrecision(6);
```

### JavaScript Number Methods

The `toPrecision()` method returns a string, with a number written with a specified length:

```
9.656
9.7
9.656
9.65600
```





Conversión de variables a números

**Number()** se puede utilizar para convertir variables de JavaScript en números:

```
Number(true);
Number(false);
Number("10");
Number(" 10");
Number("10 ");
Number(" 10 ");
Number("10.33");
Number("10,33");
Number("10 33");
Number("John");
```

### JavaScript Global Methods

The Number() method converts variables to numbers:

```
1
0
10
10
10
10
10.33
NaN
NaN
NaN
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Global Methods</h2>

<p>The Number() method converts variables to numbers:</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
  Number(true) + "<br>" +
  Number(false) + "<br>" +
  Number("10") + "<br>" +
  Number(" 10") + "<br>" +
  Number("10 ") + "<br>" +
  Number(" 10 ") + "<br>" +
  Number("10.33") + "<br>" +
  Number("10,33") + "<br>" +
  Number("10 33") + "<br>" +
  Number("John");
</script>

</body>
</html>
```



Conversión de variables a números

**parseInt()** analiza una cadena y devuelve un número entero. Se permiten espacios. Solo se devuelve el primer número:

```
parseInt("-10");
parseInt("-10.33");
parseInt("10");
parseInt("10.33");
parseInt("10 20 30");
parseInt("10 years");
parseInt("years 10");
```

### JavaScript Global Functions

#### parseInt()

The global JavaScript function parseInt() converts strings to numbers:

```
-10
-10
10
10
10
10
NaN
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Global Functions</h2>
<h2>parseInt()</h2>
<p>The global JavaScript function parseInt() converts strings to
numbers:</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
  parseInt("-10") + "<br>" +
  parseInt("-10.33") + "<br>" +
  parseInt("10") + "<br>" +
  parseInt("10.33") + "<br>" +
  parseInt("10 6") + "<br>" +
  parseInt("10 years") + "<br>" +
  parseInt("years 10");
</script>

</body>
</html>
```



Conversión de variables a números

**parseFloat()** analiza una cadena y devuelve un número. Se permiten espacios. Solo se devuelve el primer número:

```
parseFloat("10");  
parseFloat("10.33");  
parseFloat("10 20 30");  
parseFloat("10 years");  
parseFloat("years 10");
```

### JavaScript Global Methods

The `parseFloat()` method converts strings to numbers:

```
10  
10.33  
10  
10  
NaN
```

```
<!DOCTYPE html>  
<html>  
<body>  
  
<h2>JavaScript Global Methods</h2>  
  
<p>The parseFloat() method converts strings to numbers:</p>  
  
<p id="demo"></p>  
  
<script>  
document.getElementById("demo").innerHTML =  
  parseFloat("10") + "<br>" +  
  parseFloat("10.33") + "<br>" +  
  parseFloat("10 6") + "<br>" +  
  parseFloat("10 years") + "<br>" +  
  parseFloat("years 10");  
</script>  
  
</body>  
</html>
```



JavaScript MIN\_VALUE y MAX\_VALUE

**MAX\_VALUE** devuelve el mayor número posible en JavaScript.

```
let x = Number.MAX_VALUE;
```

### JavaScript Number Properties

MAX\_VALUE returns the largest possible number in JavaScript.

1.7976931348623157e+308

✓ **MIN\_VALUE** devuelve el número más bajo posible en JavaScript.

```
let x = Number.MIN_VALUE;
```

### JavaScript Number Properties

MIN\_VALUE returns the smallest number possible in JavaScript.

5e-324

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Number Properties</h2>

<p>MAX_VALUE returns the largest possible number in
JavaScript.</p>

<p id="demo"></p>

<script>
let x = Number.MAX_VALUE;
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

### Ejercicio 333 Corrige los errores de sintaxis. Instrucciones básicas

En este ejercicio vamos a repasar algunos conceptos básicos en la declaración de variables, uso de comillas y instrucciones básicas javascript.

### Ejercicio 334 ¿Qué declaraciones de variables son erróneas?

En este ejercicio vamos a aplicar el concepto de variables javascript en un documento web. Recordemos brevemente las normas para escribirlas:

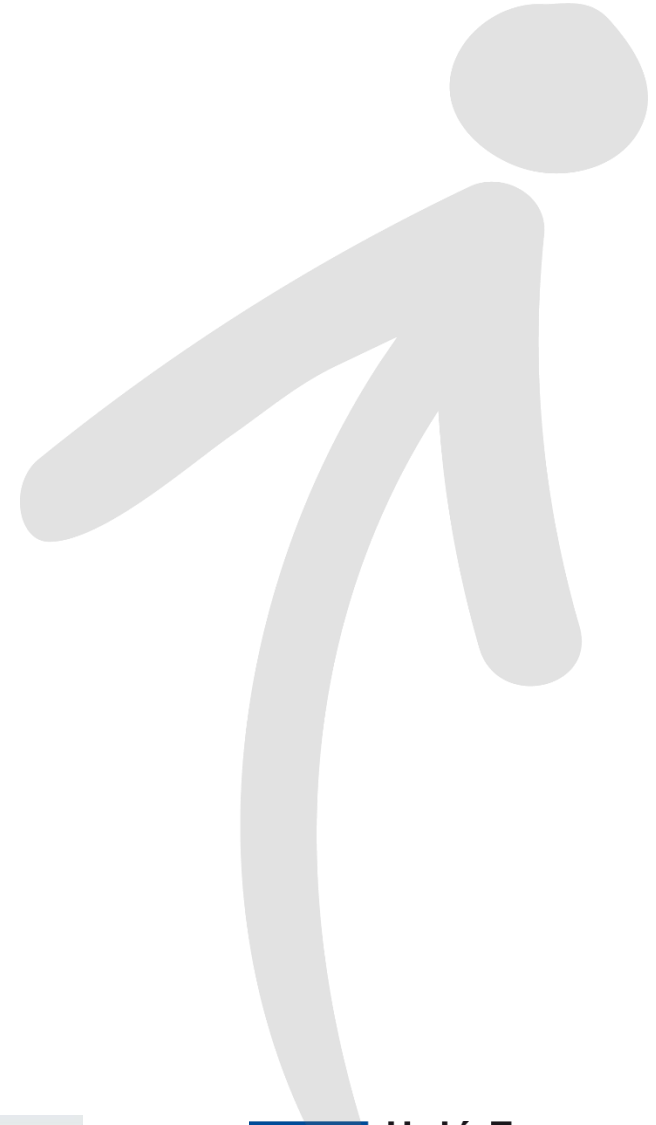
Las variables no precisan ser declaradas. Podemos usar var o no para declararlas. Se recomienda declararlas siempre pero no es obligatorio. La diferencia entre hacerlo o no la entenderéis con el resumen de clase scope de una variable. Las variables javascript son case sensitive. Por ejemplo Camello y camello son dos variables distintas. No debemos usar caracteres raros. Los únicos que están permitidos son \_ y \$. Cuidado con los espacios en blanco, puntos, porcentajes.

Los números se pueden usar. Pero nunca pueden ser el primer carácter de una variable. No debemos usar palabras reservadas para el nombre de una variable.

## 3.3. Ejercicios bloque 1

### Ejercicio 331 Acoplamiento de un script.

Modifica el documento html y desvincula el máximo posible el html y el javascript.





### **Barcelona**

Francesc Tàrraga 14  
08027 Barcelona  
93 351 78 00

### **Madrid**

Campanar 12  
28028 Madrid  
91 502 13 40

### **Reus**

Alcalde Joan Bertran 34-38  
43202 Reus  
977 31 24 36

[info@grupcief.com](mailto:info@grupcief.com)

[www.grupcief.com](http://www.grupcief.com)

