

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.



Formació



Formació  
online



Gestió de  
Bonificacions



Desenvolupament  
de Persones



Selecció



Tecnologies de  
la Informació



Millora  
Contínua



Servei de  
Prevenció Aliè



**@mihifidem**  
creativity is intelligence having fun



3

Javascript. Módulo I

3.2

**Tipos de datos.**

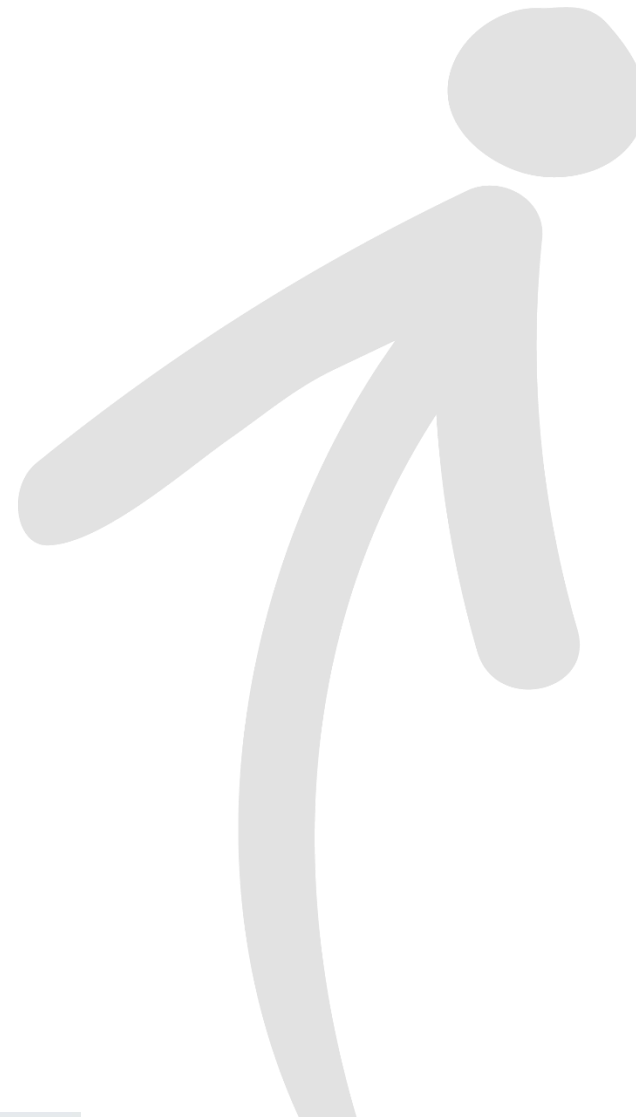
\_Datos numéricos.

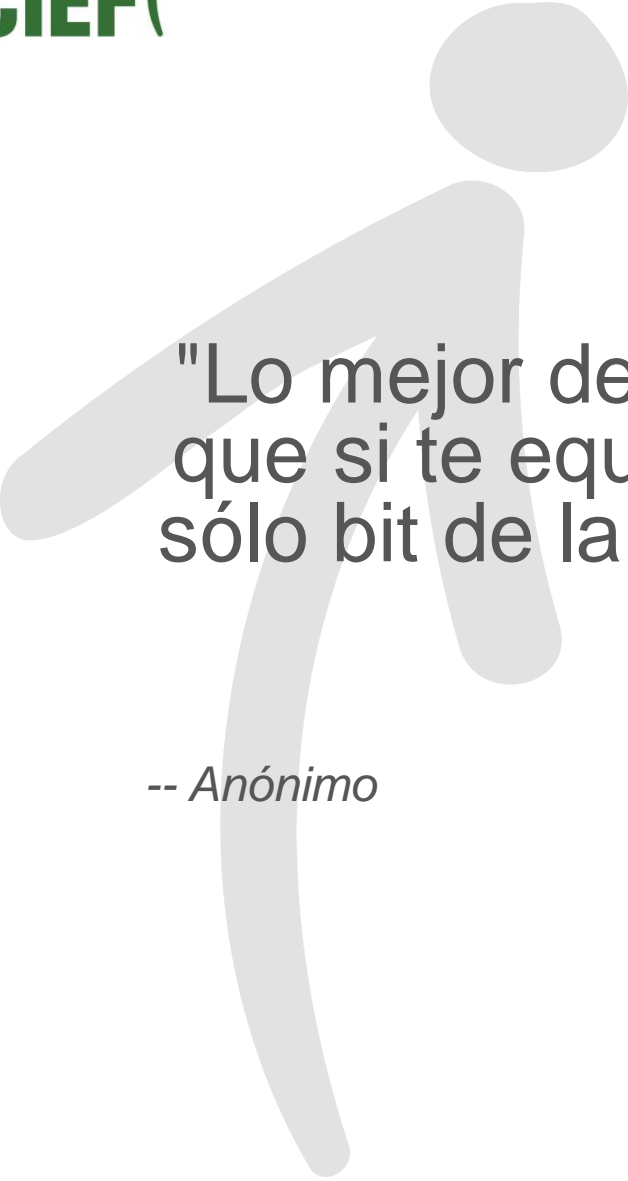
\_Datos booleanos.

\_Datos de texto.

\_Valores nulos.

\_Modo estricto





"Lo mejor de los booleanos es  
que si te equivocas estás a un  
sólo bit de la solución correcta"

-- Anónimo

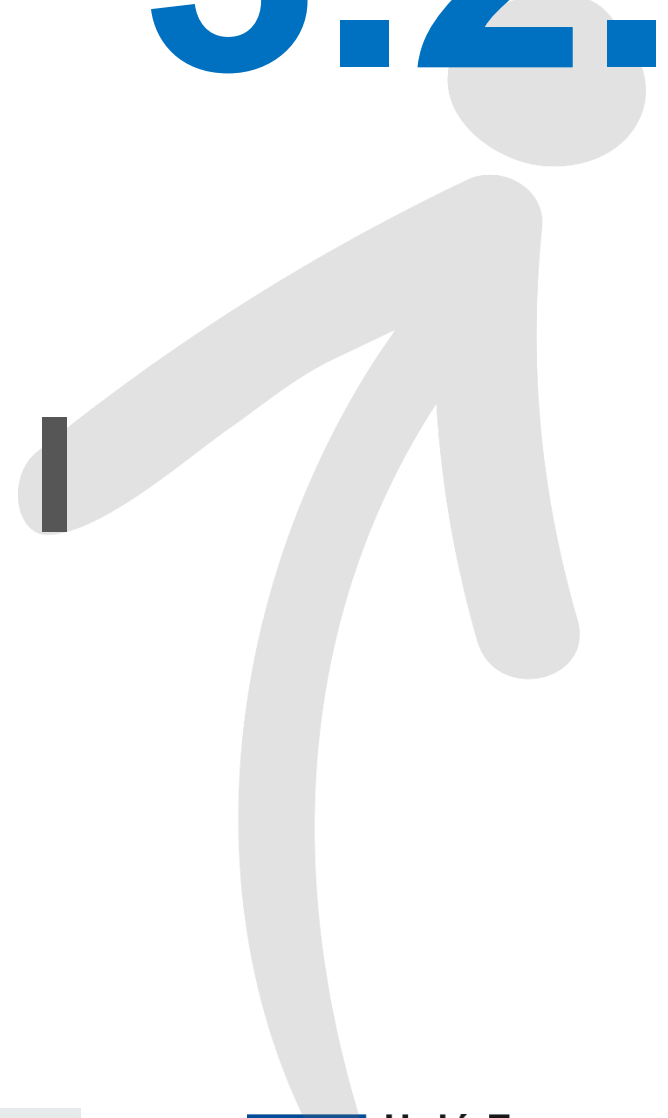


# Javascript.I



# Javascript I

## Tipos de datos



## 3.2 Tipos de datos. \_Datos numéricos

JavaScript tiene un solo tipo de número. Los números se pueden escribir con o sin decimales.

```
let x = 3.14; // A number with decimals
let y = 3;    // A number without decimals
```

Los números extra grandes o extra pequeños se pueden escribir con notación científica (exponente):

```
let x = 123e5; // 12300000
let y = 123e-5; // 0.00123
```

La aritmética de punto flotante no siempre es 100% precisa:

```
let x = 0.2 + 0.1;
```

Para resolver el problema anterior, es útil multiplicar y dividir:

```
let x = (0.2 * 10 + 0.1 * 10) / 10;
```

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

<h2>JavaScript Numbers</h2>

<p>Floating point arithmetic is not always 100% accurate.</p>

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

<script>
let x = 0.2 + 0.1;
document.getElementById("demo").innerHTML = "0.2 + 0.1 = " + x;
</script>

</body>
</html>
```

### JavaScript Numbers

Floating point arithmetic is not always 100% accurate.

0.2 + 0.1 = 0.30000000000000004



En JavaScript no se diferencia entre números enteros y reales, distinción habitual en muchos lenguajes de programación.

Para especificar un literal numérico podemos usar cualquiera de estas notaciones:

```
var num1 = 2;  
var num2 = 3.5;  
var num3 = 3.2e5;
```

El valor máximo admitido es  $1.7976931348623157e+308$  y el mínimo  $5e-324$ . i en lugar de base decimal queremos usar octal o hexadecimal, usaremos los prefijos '0' y '0x' respectivamente.

```
var enHexadecimal= 0xa3;
```

**Esto implica que al usar un número en base decimal no se pueden poner ceros a la izquierda de un numero entero.**



Si sumas dos números, el resultado será un número:

```
let x = 10;  
let y = 20;  
let z = x + y; //30
```

Si agrega dos cadenas, el resultado será una concatenación de cadenas:

```
let x = "10";  
let y = "20";  
let z = x + y; // 1020
```

Si agrega un número y una cadena, el resultado será una concatenación de cadenas:

```
let x = 10;  
let y = "20";  
let z = x + y; //1020
```

Un error común es esperar que este resultado sea 30:

```
let x = 10;  
let y = 20;  
let z = "The result is: " + x + y; //1020
```

Un error común es esperar que este resultado sea 102030:

```
let x = 10;  
let y = 20;  
let z = "30";  
let result = x + y + z; //3030
```

Las cadenas de JavaScript pueden tener contenido numérico:

```
let x = 100;    // x is a number  
  
let y = "100";  // y is a string
```

JavaScript intentará convertir cadenas en números en todas las operaciones numéricas: Esto funcionará:

```
let x = "100";  
let y = "10";  
let z = x / y; //10
```

Esto también funcionará:

```
let x = "100";  
let y = "10";  
let z = x * y; //1000
```

Y esto funcionará:

```
let x = "100";  
let y = "10";  
let z = x - y; //90
```

Pero esto no funcionará:

```
let x = "100";  
let y = "10";  
let z = x + y; //10010
```



NaN - No es un número. NaNes una palabra reservada de JavaScript que indica que un número no es un número legal.

```
let x = 100 / "Apple";
```

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

<h2>JavaScript Numbers</h2>

<p>A number divided by a non-numeric string becomes NaN (Not a
Number):</p>

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

<script>
document.getElementById("demo").innerHTML = 100 / "Apple";
</script>

</body>
</html>
```

### JavaScript Numbers

A number divided by a non-numeric string becomes NaN (Not a Number):

NaN

Sin embargo, si la cadena contiene un valor numérico, el resultado será un número:

```
let x = 100 / "10";//10
```

Puede usar la función global de JavaScript isNaN() para averiguar si un valor no es un número:

```
<script>
let x = 100 / "Apple";
document.getElementById("demo").innerHTML = isNaN(x);
</script>
```

NaNes un número: `typeof NaN` devuelve `number`:



Infinity(o -Infinity) es el valor que JavaScript devolverá si calcula un número fuera del número más grande posible.

```
let myNumber = 2;
// Execute until Infinity
while (myNumber !== Infinity) {
  myNumber = myNumber * myNumber;
}
```

### JavaScript Numbers

Infinity is returned if you calculate a number outside the largest possible number:

```
4
16
256
65536
4294967296
18446744073709552000
3.402823669209385e+38
1.157920892373162e+77
1.3407807929942597e+154
Infinity
```

//La división por 0 (cero) también genera Infinity:

```
let x = 2 / 0;
let y = -2 / 0;
```

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

<h2>JavaScript Numbers</h2>

<p>Infinity is returned if you calculate a number outside the largest
possible number:</p>

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

<script>
let myNumber = 2;
let txt = "";
while (myNumber !== Infinity) {
  myNumber = myNumber * myNumber;
  txt = txt + myNumber + "<br>";
}
document.getElementById("demo").innerHTML = txt;
</script>

</body>
</html>
```



Hexadecimal. JavaScript interpreta las constantes numéricas como hexadecimales si están precedidas por 0x.

```
let x = 0xFF;
```

### JavaScript Numbers

Numeric constants, preceded by 0x, are interpreted as hexadecimal:

0xFF = 255

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

<h2>JavaScript Numbers</h2>

<p>Numeric constants, preceded by 0x, are interpreted as
hexadecimal:</p>

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

<script>
let x = 0xFF;
document.getElementById("demo").innerHTML = "0xFF = " + x;
</script>

</body>
</html>
```



De forma predeterminada, JavaScript mostra els números com a decimals en **base 10** .  
 Però pot utilitzar el toString()mètode per generar números des de **la base 2** fins a la **base 36** .  
 El hexadecimal és **base 16** . El decimal és **base 10** . Octal és **base 8** . El binari és **base 2** .

```
let myNumber = 32;
myNumber.toString(10);
myNumber.toString(32);
myNumber.toString(16);
myNumber.toString(8);
myNumber.toString(2)
```

### JavaScript Numbers

The toString() method can output numbers from base 2 to 36:

```
32 =
Decimal 32
Hexadecimal 20
Octal 40
Binary 100000
```

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

<h2>JavaScript Numbers</h2>

<p>The toString() method can output numbers from base 2 to 36:</p>

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

<script>
let myNumber = 32;
document.getElementById("demo").innerHTML =
"32 = " + "<br>" +
" Decimal " + myNumber.toString(10) + "<br>" +
" Hexadecimal " + myNumber.toString(16) + "<br>" +
" Octal " + myNumber.toString(8) + "<br>" +
" Binary " + myNumber.toString(2);
</script>

</body>
</html>
```

Las variables lógicas sólo pueden tomar los **valores true (verdadero) y false (falso)**.

Se conocen también como **variables booleanas y su uso es muy frecuente**.

```
var numeroPar = true;  
var numeroPositivo = false;
```

true

false

## 3.2 Tipos de datos. \_Variables de texto

- Las cadenas (string) almacenan uno o más caracteres.
- Las cadenas se escriben entre comillas simples o dobles.

La posibilidad de usar los dos tipos de comillas es muy útil cuando queremos escribir una cadena que contenga comillas.

En el apartado 4.3 trataremos las cadenas de texto en profundidad.

“hola”

‘hola’



## 3.2 Tipos de datos. \_Variables de texto

Una cadena de JavaScript es cero o más caracteres escritos entre comillas.

```
let text = "John Doe";
```

Puede utilizar comillas simples o dobles:

```
let carName1 = "Volvo XC60"; // Double quotes  
let carName2 = 'Volvo XC60'; // Single quotes
```

Puede usar comillas dentro de una cadena, siempre que no coincidan con las comillas que rodean la cadena:

```
let answer1 = "It's alright";  
let answer2 = "He is called 'Johnny'";  
let answer3 = 'He is called "Johnny"';
```

## 3.2 Tipos de datos. \_Variables de texto

Una forma más segura de dividir una cadena es usar la suma de cadenas:

```
document.getElementById("demo").innerHTML = "Hello " +  
"Dolly!";
```

Puede utilizar comillas simples o dobles:

```
let carName1 = "Volvo XC60"; // Double quotes  
let carName2 = 'Volvo XC60'; // Single quotes
```

Puede usar comillas dentro de una cadena, siempre que no coincidan con las comillas que rodean la cadena:

```
let answer1 = "It's alright";  
let answer2 = "He is called 'Johnny'";  
let answer3 = 'He is called "Johnny"';
```



### Escape Character

Debido a que las cadenas deben escribirse entre comillas, JavaScript malinterpretará esta cadena:

```
let text = "We are the so-called "Vikings" from the north.";
```

La cadena se cortará a "Somos los llamados".

La solución para evitar este problema es utilizar el **carácter de escape de barra invertida**.

El carácter de escape de barra invertida ( \ ) convierte los caracteres especiales en caracteres de cadena:

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

<h2>JavaScript Strings</h2>

<p>The escape sequence \" inserts a double quote in a string.</p>

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

<script>
let text = "We are the so-called \"Vikings\" from the north.";
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

Code	Result	Description
\'	'	Single quote
\"	"	Double quote
\\	\	Backslash

La secuencia \" inserta una comilla doble en una cadena:

### JavaScript Strings

The escape sequence \" inserts a double quote in a string.

We are the so-called "Vikings" from the north.



```
let text= 'It\'s alright.';
```

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

<h2>JavaScript Strings</h2>

<p>The escape sequence \' inserts a single quote in a string.</p>

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

<script>
let text = 'It\'s alright.';
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

### JavaScript Strings

The escape sequence \' inserts a single quote in a string.

It's alright.

La secuencia \\ inserta una barra invertida en una cadena:

```
let text = "The character \\ is called backslash.";
```

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

<h2>JavaScript Strings</h2>

<p>The escape sequence \\ inserts a backslash in a string.</p>

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

<script>
let text = "The character \\ is called backslash.";
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

### JavaScript Strings

The escape sequence \\ inserts a backslash in a string.

The character \ is called backslash.

## 3.2 Tipos de datos. \_Variables. Undefined y null

**Cuando se crea una variable sin asignarle valor, su valor es undefined (no definido).**

**Con null se puede indicar que la variable está vacía.**

Distinguir entre ambos puede ser un poco complicado, pero iremos viendo ejemplos de uso a lo largo del manual.

sloppy mode — modo poco riguroso al modo no estricto predeterminado. Este no es un término oficial, pero tenlo en cuenta, por si acaso.

El modo estricto de [ECMAScript 5](#) es una forma de elegir una variante *restringida* de *JavaScript*, así implícitamente se deja de lado el modo poco riguroso. El modo estricto no es sólo un subconjunto: *intencionalmente* tiene diferencia semántica del código normal. Los navegadores que no admiten el modo estricto ejecutarán el código con un comportamiento diferente a los que sí lo soportan, por lo tanto no confíes en el modo estricto sin antes hacer pruebas de sus características más relevantes. Los modos estricto y no estricto pueden coexistir, por lo tanto el código se puede transformar a modo estricto incrementalmente.

El modo estricto tiene varios cambios en la semántica normal de JavaScript:

- Elimina algunos errores silenciosos de JavaScript cambiándolos para que lancen errores.
- Corrige errores que hacen difícil para los motores de JavaScript realizar optimizaciones: a veces, el código en modo estricto puede correr más rápido que un código idéntico pero no estricto.
- Prohíbe cierta sintaxis que probablemente sea definida en futuras versiones de ECMAScript.

**'use strict'**

pais = "Holanda"; //No me deja hacerlo

var pais = "Holanda"; //Si me deja



### **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)**

