

Intro JavaScript: no es Java





JAVASCRIPT?



<SCRIPT>ALERT('PFFFT')</SCRIPT>

imgflip.com



Qué es javascript?

JavaScript es el lenguaje de programación que debes usar para añadir características interactivas a tu sitio web, (por ejemplo, juegos, eventos que ocurren cuando los botones son presionados o los datos son introducidos en los formularios, efectos de estilo dinámicos, animación, y mucho más).



JavaScript Code:

```
1 | function helloSyntaxHighlighter()  
2 | {  
3 |     return "hi!";  
4 | }
```

Java Code:

```
1 | public class JavaClass {  
2 |     public static void main(String[] args) {  
3 |         System.out.println("Hello World!");  
4 |     }  
5 | }
```

No se dejen engañar, JavaScript y Java no son lo mismo!



Un poco de historia

A principios de los años 90, la mayoría de usuarios que se conectaban a Internet lo hacían con módems a una velocidad máxima de 28.8 kbps. En esa época, empezaban a desarrollarse las primeras aplicaciones web y por tanto, las páginas web comenzaban a incluir formularios complejos.

Con unas aplicaciones web cada vez más complejas y una velocidad de navegación tan lenta, surgió la necesidad de un lenguaje de programación que se ejecutara en el navegador del usuario. De esta forma, si el usuario no rellenaba correctamente un formulario, no se le hacía esperar mucho tiempo hasta que el servidor volviera a mostrar el formulario indicando los errores existentes.

Brendan Eich, un programador que trabajaba en Netscape, pensó que podría solucionar este problema adaptando otras tecnologías existentes (como *ScriptEase*) al navegador Netscape Navigator 2.0, que iba a lanzarse en 1995. Inicialmente, Eich denominó a su lenguaje *LiveScript*.



Posteriormente, Netscape firmó una alianza con Sun Microsystems para el desarrollo del nuevo lenguaje de programación. Además, justo antes del lanzamiento Netscape decidió cambiar el nombre por el de JavaScript. La razón del cambio de nombre fue exclusivamente por marketing, ya que Java era la palabra de moda en el mundo informático y de Internet de la época.

La primera versión de JavaScript fue un completo éxito y Netscape Navigator 3.0 ya incorporaba la siguiente versión del lenguaje, la versión 1.1. Al mismo tiempo, Microsoft lanzó JScript con su navegador Internet Explorer 3. JScript era una copia de JavaScript al que le cambiaron el nombre para evitar problemas legales.

Para evitar una guerra de tecnologías, Netscape decidió que lo mejor sería estandarizar el lenguaje JavaScript. De esta forma, en 1997 se envió la especificación JavaScript 1.1 al organismo ECMA (*European Computer Manufacturers Association*).

ECMA creó el comité TC39 con el objetivo de "*estandarizar de un lenguaje de script multiplataforma e independiente de cualquier empresa*". El primer estándar que creó el comité TC39 se denominó ECMA-262, en el que se definió por primera vez el lenguaje ECMAScript.

Por este motivo, algunos programadores prefieren la denominación *ECMAScript* para referirse al lenguaje JavaScript. De hecho, JavaScript no es más que la implementación que realizó la empresa Netscape del estándar ECMAScript.



Variables

Alcances (let, var & const)

Las **variables** son contenedores en los que se pueden almacenar valores. Primero debemos declarar el tipo de la variable mediante una palabra clave (que puede ser **let**, **const** o **var**... var prohibido usarla!), seguida del nombre que vamos a darle a dicha variable.



```
1  var oracion = 'Soy una oración';  
2  
3  let numero = 0;  
4  
5  const verdadero = true;
```



Unas notas importantes, antes de profundizar en los tipos de variables.

- Todas las líneas en JS **deben acabar en punto y coma (;)** para indicar que es justo allí donde termina la declaración. De no incluirlos, se pueden obtener resultados inesperados. Sin embargo, algunas personas creen que esta convención es sólo parte de una buena práctica. Hay otras reglas sobre cuándo se debe y no se debe usar punto y coma.
- Se puede llamar a una variable con cualquier nombre, pero **hay restricciones** (no se pueden utilizar palabras reservadas). Los nombres de las variables deben ser intuitivas y dejar claro qué están almacenando.
- JavaScript distingue entre mayúsculas y minúsculas. **miVariable** es una variable distinta a **mivariable**.
- Se utiliza la convención de **camelCase** para nombrar a las variables.



¿Cuándo necesitamos utilizar punto y coma?

Es **obligatorio**:

Cuando dos o más declaraciones están en la misma línea.

En el ejemplo, el punto y coma es **obligatorio** entre el 0 y el incremental (i++), pero es **opcional** al final de la declaración.



Es **opcional**:

Es opcional después de cada declaración. El punto y coma en JavaScript se utiliza para separar declaraciones, pero se puede omitir si la declaración va seguida de un salto de línea (o hay una declaración en un {bloque}).

Una **declaración** es un fragmento de código que le dice a la computadora que haga algo. Estos tipos de declaraciones son muy comunes:



- **Declaración de variable:**
`let i;`
- **Asignación de valor:**
`i = 5;`
- **Asignación de valor:**
`i = i + 1;`
- **Asignación de valor** (con el mismo valor que la variable de arriba):
`i ++;`
- **Declaración y asignación:**
`let x = 9;`
- **Declaración, asignación y función:**
`let fun = function() { ... };`
- **Llamado de una función:**
`alert("hola");`



Todas estas declaraciones pueden terminar con un punto y coma, pero ninguno debe hacerlo. Algunos consideran que es un buen hábito terminar cada declaración con un (;). Eso hace que el código sea un poco más fácil de analizar y comprimir: si se llegaran a eliminar los saltos de línea, no necesitaríamos preocuparnos de problemas ya que contaríamos con la contención y la división de los punto y coma.

¿Cuándo deberíamos evitarlo?

1. Después del cierre de una llave:

No debe ponerse punto y coma después del cierre de una llave (}). Las únicas excepciones son las declaraciones de asignación, como:

```
let obj = {};
```

- **No necesitamos (;) después de:**

```
if (...) {...} else {...}  
for (...) {...}  
while (...) {...}
```



- **Sí debemos usar (;) cuando:**

do {...} while (...);

- **No debemos usar (;) al finalizar una función:**

```
function functionName(arg) {  
    ... do this ...  
}
```

2. Después de los paréntesis en una declaración if, for, while o switch:

```
if(0 === 1); { alert("hi") }
```

Es equivalente a:

```
if(0 === 1);  
alert("hi");
```



El código anterior disparará el alert, aunque la condición 0 “es igual a” 1 no se cumpla (porque ya no sería parte de esa condición).



Pero ojo, **que hay una excepción:**

Dentro del () de un bucle for, los puntos y coma sólo van después de la primera y segunda declaración; nunca después de la tercera.

for (let i = 0; i < 10; i++) { ... acciones ... }

Forma correcta

for (let i = 0; i < 10; i++;) { ... acciones ... }

Forma incorrecta



Restricciones al nombrar variables

- Una variable **debe iniciar siempre con una letra** o un **guión bajo** (_).
- Una variable **puede contener números solamente después de la primer letra**.
- No está permitido dejar un espacio en blanco a lo largo de la variable.
- Aunque las variables pueden tener el largo que queramos, lo recomendable son variables cortas (**entre 20 y 30 caracteres como máximo**).
- No podemos utilizar las **palabras reservadas** para nombrar una variable.

Palabras reservadas en JavaScript:

await	break	case	catch	class	const	continue	debugger
default	delete	do	else	export	extends	finally	for
if	import	in	instanceof	new	return		super
switch	this	throw	try	typeof	var	void	while
with	yield	let	static	null	false	true	enum
implements		interface		package		private	protected
public	abstract	boolean	byte	char	double	final	float
goto	int	long	native	short	synchronized		
transient							



Ahora sí...

Declaración de variables:

- **var:** variables globales a las cuales podemos acceder desde cualquier parte de nuestro código.
- **let:** variables limitadas al alcance del bloque en donde son definidas.
- **const:** variables inmutables, cuyo valor no se puede modificar.

¿En qué se distinguen? Desarrollo en variables.js



Variables

Tipos de variables

- **String**
- **Number**
- **Boolean**
- **Object**
- **Array**
- **Function**

El tipo de variable dependerá del tipo de valor que almacena; empezaremos profundizando acerca de los **valores de tipo string**.



String value

Este tipo se utiliza para representar y manipular una secuencia de caracteres.

Descripción:

Las cadenas son útiles para almacenar datos que se pueden representar en forma de texto. Existen muchos **métodos** para aplicar sobre ellas.

Primero debemos tener claro la diferencia entre los **tipos de cadenas primitivas** y los **objetos String()**.



```
1 let typeOne = 'Esta es un tipo de cadena primitiva, entre comillas simples.'  
2 let typeTwo = "Esta es un tipo de cadena primitiva, entre comillas dobles."  
3 let typeThree = `Esta es un tipo de cadena primitiva, entre comillas invertidas.`  
4  
5 let typeFour = new String('Este es un objeto String')
```



Podemos hacer muchas cosas sobre los strings, como:

- **Acceder a un caracter**

Existen dos formas de acceder a un caracter individual en una cadena.

La primera con el método **charAt()**:



```
1 let cat = 'Gato'
2 let g = cat.charAt(0);
3 console.log(g) //devuelve G
```



La segunda forma es una modalidad introducida en ECMAScript 5, que se trata de tomar a la cadena de texto como un objeto similar a un array (arreglo), donde **los caracteres individuales corresponden a un índice numérico**:



```
1 let cat = 'Gato'
2 let g = cat[0] // Donde G = 0; a = 1; t = 2; o = 3
3 console.log(g) // Devuelve G
```

Nota importante: en JavaScript se comienza a contar a partir del 0.



- **Concatenar cadenas de texto**

Muchas veces necesitamos concatenar distintas variables con cadenas de texto. Existen al menos tres formas:

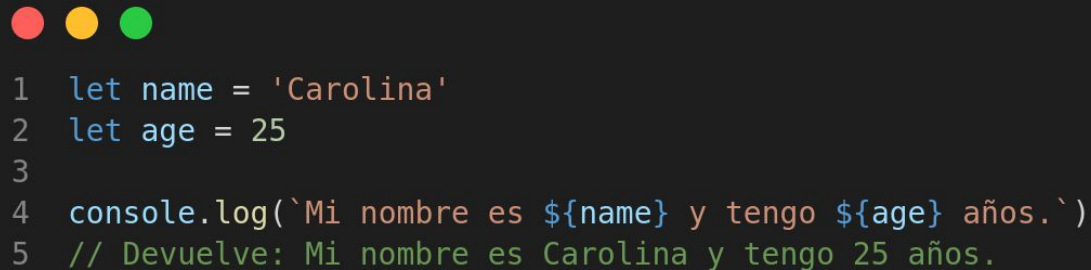
1. La primera, a través de **(+)**:



```
1 let name = 'Carolina'
2 let age = 25
3
4 console.log('Mi nombre es ' + name + ' y tengo ' + age + ' años.')
5 // Devuelve: Mi nombre es Carolina y tengo 25 años.
```



2. A través de la **interpolación de texto**, con comillas invertidas:




```
1 let name = 'Carolina'
2 let age = 25
3
4 console.log(`Mi nombre es ${name} y tengo ${age} años.`)
5 // Devuelve: Mi nombre es Carolina y tengo 25 años.
```

Esta es la manera más recomendada, ya que el código queda más limpio.



3. Con el método **concat()**:




```
1 let name = 'Mi nombre es Carolina'
2 let age = ' y tengo 25 años'
3 let conc = name.concat(age)
4 console.log(conc)
5 // Devuelve: Mi nombre es Carolina y tengo 25 años
```

Este método resulta extraño, porque nos obliga a convertir la variable `age` (que antes era de tipo `number`), a una de tipo `string` para que la oración tenga sentido.



- **Comparar cadenas**

En JavaScript, para comparar cadenas, se utilizan los operadores **“menor que”** (<) y **“mayor que”** (>).



```
1 // Tenemos las variables a y b:
2 let a = 'a'
3 let b = 'b'
4
5 // Comparamos:
6 if (a < b) {
7     console.log(`${a} es menor que ${b}.`)
8 } else if (a > b) {
9     console.log(`${a} es mayor que ${b}.`)
10 } else {
11     console.log(`${a} y ${b} son iguales.`)
12 }
```

¿Qué creen que retorna?



El anterior script devuelve “a es menor que b”, ya que “a” toma una posición [0] en el abecedario, y “b” toma una posición [1].

¿Qué pasa con las mayúsculas?

```
1 // Tenemos las variables a y b:
2 let a = 'a'
3 let b = 'B'
4
5 // Comparamos:
6 if (a < b) {
7     console.log(`${a} es menor que ${b}.`)
8 } else if (a > b) {
9     console.log(`${a} es mayor que ${b}.`)
10 } else {
11     console.log(`${a} y ${b} son iguales.`)
12 }
```

¿Qué piensan que retorna?



El anterior script retorna que "a" es mayor que "B" porque JavaScript es **case sensitive**, por lo que primero recorre las mayúsculas y luego las minúsculas, con lo que las mayus tendrán valores menores que las minus.

El temita con los strings primitivos y los objetos String:



```
1 let name = 'Carolina'
2 let nameObject = new String(name)
3
4 console.log(typeof name) // Devuelve tipo string
5 console.log(typeof nameObject) // Devuelve tipo object
```



Por ende, **tenemos que tener cuidado qué métodos aplicamos a un string primitivo o a un objeto de string:**



```
1 let s1 = '2 + 2'
2 console.log(s1) // Retorna 2 + 2 porque los strings no se pueden sumar
3
4 let s2 = new String('2 + 2')
5 console.log(s2) // Retorna 2 + 2 porque los strings no se pueden sumar
```



```
1 // Aplicamos un método eval para poder sumarlos:
2
3 console.log(eval(s1)) // Devuelve el número 4
4 console.log(eval(s2)) // Devuelve la cadena 2 + 2
```





```
1 let s3 = eval(s1)
2 let s4 = eval(s2)
3
4 console.log(typeof s3) // Devuelve que s3 es de tipo number
5 console.log(typeof s4) // Devuelve que s4 es de tipo object
```

Como podemos observar, **la variable s2 no ha cambiado su tipo object** en ningún momento (ni en s2 ni en s4, a pesar de aplicarse sobre ella el método eval).

En cambio, la variable **s3**, al aplicarle el método eval sobre la variable **s1** (a pesar de ser un string), define su valor a tipo number.

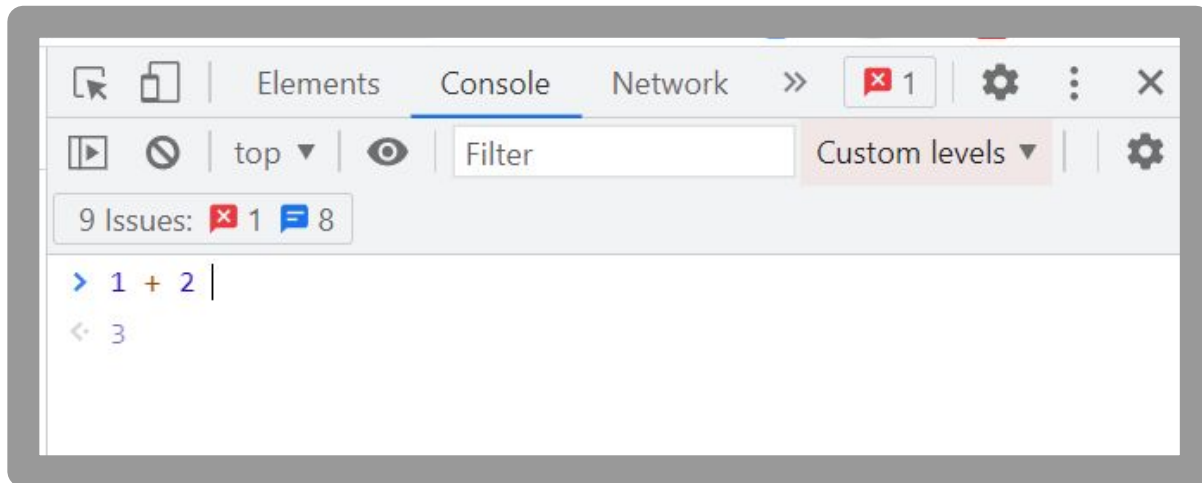
No es la variable s1 quien cambia su tipo: su tipo continúa siendo string.



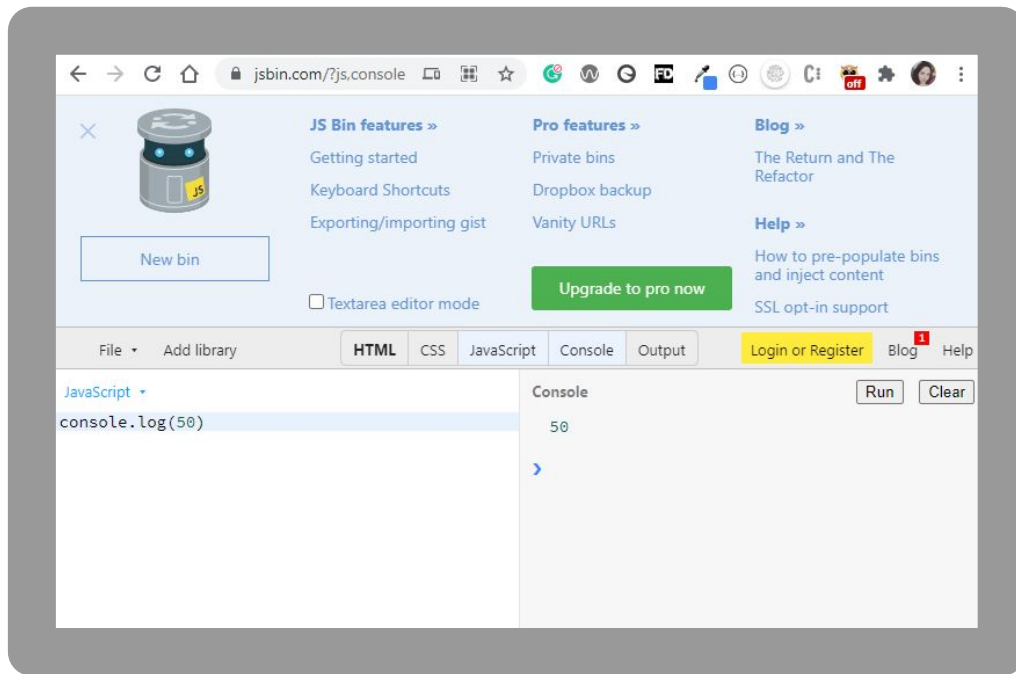
Cómo corremos JavaScript?

Ahora ya no podemos simplemente usar el live server para ver nuestro código, porque...
qué esqueleto tenemos? Tenemos varias opciones entonces:

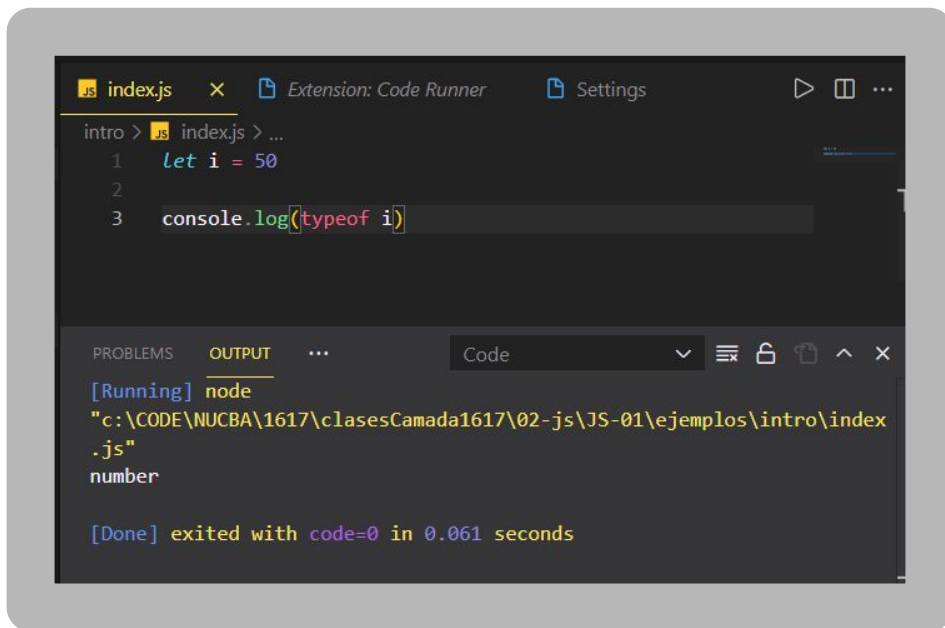
- la consola del inspeccionador:



- otra opción es usar algún sitio como estos:
 - <https://jsfiddle.net/>
 - <https://jsbin.com/?js,console>: este en lo personal es mi favorito.
 - <https://playcode.io/new/>



- la tercera opción es desde el code, y recibir en la consola los resultados: para eso instalamos code runner (<https://marketplace.visualstudio.com/items?itemName=formulahendry.code-runner>), nos aseguramos que esté en tomando javaScript como lenguaje, y hacemos click al botoncito de play que nos va a aparecer arriba.



The screenshot shows the Visual Studio Code interface with the Code Runner extension installed. The editor displays a file named `index.js` with the following code:

```
intro > .js index.js > ...  
1 let i = 50  
2  
3 console.log(typeof i)
```

The bottom panel shows the Output window with the following text:

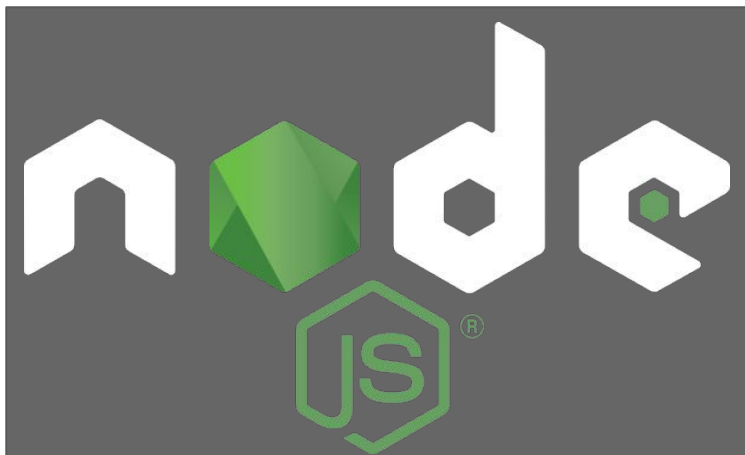
```
[Running] node  
"c:\CODE\NUCBA\1617\clasesCamada1617\02-js\JS-01\ejemplos\intro\index  
.js"  
number  
  
[Done] exited with code=0 in 0.061 seconds
```



- por último, y con esto ya nos preparamos para lo que va a venir, podemos usar node para ejecutarlo.

Vamos a descargar la versión LTS e instalarla: <https://nodejs.org/es/>

Y después simplemente en la consola es escribir **node {nombreDelArchivo}** para ejecutarlo!



Non-zero value



null



0



undefined





Ejercicios



Vamos a declarar:

- Variables de tipo **string**:
 - Name
 - LastName
 - Country
 - Description
- Variables de tipo **number**:
 - Age

Debemos hacer un **console log** que retorne:

*Hola, soy **name lastName**, tengo **age** y vivo en **country**. Me considero **description** y estudio en NUCBA para romperla mañana como programador/a.*

Importante

Realicen la **concatenación de texto** de las dos formas que vimos anteriormente (sin el concat), para que puedan practicarlas.

NUCBA

