

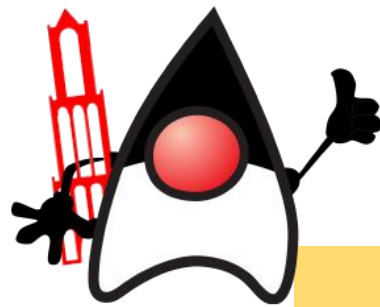
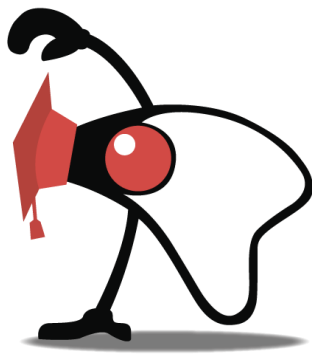


<codoa
codo/>



Curso FullStack Python

Codo a Codo 4.0



JavaScript

Parte 1



¿Qué es JavaScript?

JavaScript es un **lenguaje de programación**, o lo que es lo mismo, un mecanismo con el que podemos decirle a nuestro navegador qué tareas debe realizar, en qué orden y cuántas veces, por ejemplo. Con JS vamos a agregarle **comportamiento** a nuestro sitio y lograr que el usuario **interactúe** con él.

Javascript rompe con lo estático del HTML y permite crear elementos dinámicos e interactivos, mejorando ampliamente la interacción de los usuarios con una página web.

Javascript es la tercera pieza fundamental del desarrollo web

Más info: <https://lenguajejs.com/javascript/introduccion/que-es-javascript/>

ECMAScript

ECMAScript es el estándar que a partir del año 2015 a la actualidad se encarga de regir como debe ser interpretado y funcionar el lenguaje JavaScript, siendo este (JS – JavaScript) interpretado y procesado por multitud de plataformas, entre las que se encuentran los navegadores web.

Más info: <https://openwebinars.net/blog/que-es-ecmascript/>

Versiones de ECMAScript

A lo largo de los años, Javascript ha ido sufriendo modificaciones que los navegadores han ido implementando para acomodarse a la última versión de ECMAScript cuanto antes. La lista de versiones de ECMAScript aparecidas hasta el momento son las siguientes, donde encontramos las versiones enmarcadas en lo que podemos considerar el pasado de Javascript:

Ed.	Fecha	Nombre formal / informal	Cambios significativos
1	JUN/1997	ECMAScript 1997 (ES1)	Primera edición
5	DIC/2009	ECMAScript 2009 (ES5)	Strict mode, JSON, etc...
6	JUN/2015	ECMAScript 2015 (ES6)	Clases, módulos, generadores, hashmaps, sets, for of, proxies...
7	JUN/2016	ECMAScript 2016	Array includes(), Exponenciación **
8	JUN/2017	ECMAScript 2017	Async/await
9	JUN/2018	ECMAScript 2018	Rest/Spread operator, Promise.finally()...
10	JUN/2019	ECMAScript 2019	Flat functions, trimStart(), errores opcionales en catch...
11	JUN/2020	ECMAScript 2020	Dynamic imports, BigInt, Promise.allSettled

Características de JavaScript



Lenguaje del lado del cliente: Significa que **se ejecuta en la máquina del propio cliente** a través de un navegador. Algunos de estos lenguajes son el propio JavaScript, HTML, CSS.

Lenguaje orientado a objetos: **Utiliza clases y objetos como estructuras** que permiten organizarse de forma simple y son reutilizables durante todo el desarrollo. Otros lenguajes orientados a objetos son Java, Python o C++.

De tipado débil o no tipado: **No es necesario especificar el tipo de dato** al declarar una variable. Esta característica supone una gran ventaja a la hora de ganar rapidez programando, pero puede provocar que cometamos más errores que si tuviéramos esa restricción que poseen los lenguajes de tipado fuerte como C++ o Java.

De alto nivel: Su sintaxis es fácilmente comprensible por su similitud al lenguaje de las personas. Se le llama de “alto nivel” porque **su sintaxis se encuentra alejada del nivel máquina**, es decir, del código que procesa una computadora para ejecutar lo que nosotros programamos.

Un lenguaje de alto nivel como Javascript permite que su barrera de entrada y su curva de aprendizaje se acorte drásticamente. Un ejemplo podría ser que la sentencia condicional empiece por “IF” que significa “si...” en inglés, permitiendo asociar rápidamente su funcionamiento y significado. Otro lenguaje de alto nivel muy utilizado y uno de los mejores para iniciarse en programación por esta característica es Python.

Lenguaje interpretado: Utiliza un intérprete que **permite convertir las líneas de código en el lenguaje de la máquina**. Esto tiene un gran número de ventajas como la reducción del procesamiento en servidores web al ejecutarse directamente en el navegador del usuario, o que es apto para múltiples plataformas permitiendo usar el mismo código. Además de JS, otros ejemplos de lenguajes interpretados son C#, Ruby, Java o Python.

Muy utilizado por desarrolladores: A la hora de elegir si aprender o no un nuevo lenguaje, no sólo hay que informarse sobre el tipo de lenguaje o su curva de aprendizaje, si no también su demanda en el mercado. Javascript es en la actualidad uno de los lenguajes más demandados de los últimos años por su versatilidad y su infinita capacidad para crear plataformas cada vez más atractivas.



¿Qué ventajas tiene trabajar sobre JS?

- Trabajamos sobre el cliente, no vamos al servidor, sino que estas instrucciones corren en el navegador del cliente, del usuario, en este caso Chrome. Yo no le estoy diciendo desde un servidor que la página Web consulta *“mostrale por pantalla tal texto”*, que también se puede hacer.
- De esta manera me **ahorro el tiempo** que tarda en viajar la información, hacer la solicitud al servidor, el servidor responderle y volver a la PC del cliente a través de distintos dispositivos de conexión.
- Si quisiera validar los datos de un formulario de una aplicación, por ejemplo un mail (debe tener un @, después del @ debe tener al menos un punto, tendría que tener alguna extensión, con dos o tres caracteres o ahora cuatro si es .info) ya no hace falta que vaya a la base de datos, al servidor para chequearlo, sino que lo puedo hacer directamente desde el navegador del cliente ahorrándome todos estos tiempos de respuesta y los recursos del servidor

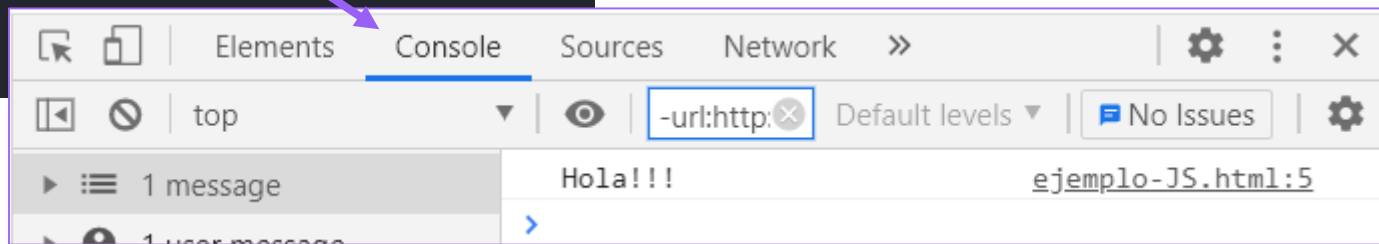
Cada vez le pedimos más al navegador para ahorrar recursos del servidor y que tenga un tiempo de respuesta más rápido, pero cada vez dependemos más que el cliente tenga una mejor conexión, un mejor procesador, un navegador actualizados, etc.

Comenzando con JavaScript

Una forma sería agregar la etiqueta **<script>** en la etiqueta **<head>**, utilizando una referencia **interna**.

```
<html>
  <head>
    <title>Titulo de la pagina</title>
    <script>
      console.log("Hola!!!")
    </script>
  </head>
  <body>
    <p>Ejemplo de texto.</p>
  </body>
</html>
```

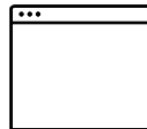
Desde la consola del navegador podremos ver que se escribe el texto "Hola!!!", esto se logra a través de la instrucción **console.log()**



La ubicación de la etiqueta Script

De acuerdo a donde esté ubicada la etiqueta **<script>** la página web tomará diferentes estados:

- **En <head>**: Se descargará el archivo JavaScript **antes** de empezar a dibujar la página (*cuando la página está en blanco*).
- **En <body>**: Se descargará el archivo JavaScript **durante** el dibujado de la página (*cuando la página se haya dibujada hasta el <script>*).
- **Antes de </body>**: Se descargará el archivo JavaScript **después** de dibujar la página (*cuando la página se haya dibujado en su totalidad*).

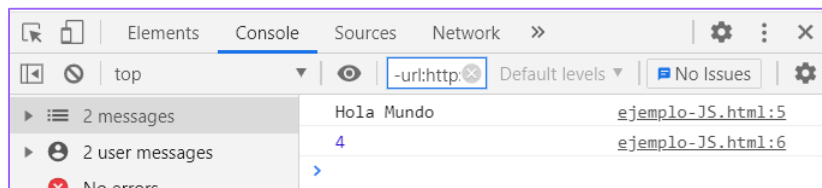


La consola de JavaScript

El clásico primer ejemplo cuando se comienza a programar es crear un programa que muestre por pantalla un texto, generalmente el texto «Hola Mundo». También podemos hacer operaciones numéricas:

```
console.log("Hola Mundo");  
console.log(2 + 2);
```

JS



La consola posee otras funciones...

Función	Descripción
console.log()	Muestra la información proporcionada en la consola Javascript.
console.info()	Equivalente al anterior. Se utiliza para mensajes de información.
console.warn()	Muestra información de advertencia. Aparece en amarillo.
console.error()	Muestra información de error. Aparece en rojo.
console.clear()	Limpia la consola. Equivalente a pulsar CTRL + L o escribir <code>clear()</code> .



La consola de JavaScript

Para acceder a la consola Javascript del navegador, podemos pulsar **CTRL+SHIFT+I** sobre la pestaña de la página web en cuestión, lo que nos llevará al **Inspector de elementos** del navegador, que es un panel de control general donde podemos ver varios aspectos de la página en la que nos encontramos: su etiquetado HTML, sus estilos CSS, etc...

Nos moveremos a la pestaña **Console** y ya nos encontraremos en la **consola Javascript** de la página.

También se puede utilizar directamente el atajo de teclado **CTRL+SHIFT+J**, que en algunos navegadores nos lleva directamente a la consola.

Mostrando datos...

Podemos mostrar texto separados por comas...

```
console.log("¡Hola a todos! Observen este número: ", 5 + 18); JS
```

```
¡Hola a todos! Observen este número: 23 ejemplo-JS.html:5
```

En esta consola, podemos escribir funciones o sentencias de Javascript que estarán actuando en la página que se encuentra en la pestaña actual del navegador. De esta forma podremos observar los resultados que nos devuelve en la consola al realizar diferentes acciones.

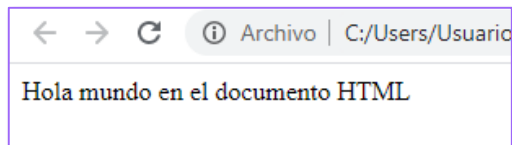
Uso de document.write()

Con **document.write()** podemos trabajar directamente dentro del propio documento HTML:

```
document.write("Hola mundo en el documento HTML");
```

JS

*En este caso lo que hacemos es imprimir en el body (que en el HTML no tiene nada) un determinado texto. Esto sucede porque el documento HTML está vinculado a través de un script con el archivo de JS que está en una determinada ubicación. Al documento HTML que está invocando le dice que **escriba** (write) un determinado texto en el body.*



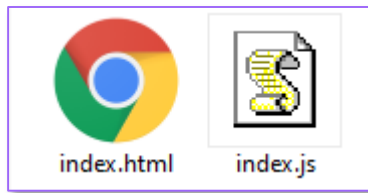
De esta manera ya no tendremos una estructura tan estática, sino que ya la voy a poder “controlar un poco más” desde JS. En este caso estoy controlando el contenido que estoy mostrando y ya no es un texto fijo que tengo que ir al HTML a modificarlo, sino que podría ser modificado desde el código de JS.

Incorporando un archivo externo

Al igual que con CSS, donde teníamos un archivo externo de extensión **.css** que vinculábamos con nuestro documento HTML en JavaScript podemos hacer lo mismo, vinculando el archivo con extensión **.js** al documento dentro de la etiqueta **<script>**, sólo que en este caso, haremos referencia al archivo Javascript con un atributo **src** (source):

```
<html>
  <head>
    <title>Título de la página</title>
    <script src="index.js"></script>
  </head>
  <body>
    <p>Ejemplo de texto.</p>
  </body>
</html>
```

HTML



Este tipo de archivos se suelen incorporar en una **carpeta** llamada "js" con archivos de extensión .js para indicar que son archivos de JavaScript.

```
console.log("Bienvenidos a Codo a Codo");
```

JS

Bienvenidos a Codo a Codo

Comentarios

Los comentarios son utilizados por los programadores para anotaciones, no son tenidos en cuenta por el navegador.

Comentarios de línea

```
// esto es un comentario de línea
```

Comentarios de bloque

```
/*  
esto es un comentario de bloque (multilínea)  
*/
```

Al no ser tenidos en cuenta por el navegador, son un buen recurso cuando queremos omitir la ejecución de ciertas instrucciones.

Para colocar/quitar comentarios en VSC podemos utilizar el atajo CTRL + ⌘

*Cuando escribamos comentarios no debemos escribir **qué** hacemos, sino **por qué** lo hacemos.*

Variables

- Es el nombre genérico que se le da a **pequeños espacios de memoria** donde se guarda una dato determinado, de forma muy similar a las incógnitas en matemáticas.
- Un programa puede tener **muchas variables**, y cada una de ellas tendrá un nombre que la identifique, un valor y un tipo de dato.
- El nombre se utiliza para diferenciarlas unas de otras y hacer referencia a ellas, el valor es la información que contienen y el tipo de dato es la naturaleza de ese valor.
- Se llaman variables porque podemos cambiarle su valor a lo largo del programa, según necesitemos.
- La idea del uso de una variable es que se puede reutilizarla varias veces a lo largo del programa.



Imaginemos a la variable como una “cajita” dentro de nuestro programa, que tiene tres características:

- **Su nombre:** Debe ser representativo de la información que contiene
- **El tipo de datos:** Podrá ser número, texto, valores booleanos, etc.
- **Su contenido:** Asociado al tipo de datos

En nuestro caso la variable Var tiene un tipo de dato numérico de valor 5.

Variables

- Los nombres de las variables son para distinguir una de otras, pero además deben tener un **valor** asociado ya que, si yo nombro variables a, b, c solas no tenemos idea de qué representan.
- Hay caracteres como el guion medio que no se suele utilizar, tampoco el % ni el ?. El signo \$ se reserva para algunas referencias particulares. Los caracteres con acento o la ñ no se suelen utilizar porque son propios del lenguaje español y eso no lo utilizamos en los nombres de variables como una buena práctica de programación.
- Se recomienda usar la escritura camelCase en el nombre de variables que tendrán más de una palabra.
- Si tengo un cálculo aritmético, por ejemplo, **num1 = 1+4** se calcula lo que está a la derecha, primero se hace el cálculo aritmético, y luego se asigna a izquierda.

Una variable será **indefinida** si es declarada con **var**, pero no se le asigna un valor (`var num3;`). Una cosa es declararla y decirle “esto va a ser una variable que va a tener el nombre num3” y otra cosa es asignarle un valor. En este caso la variable está “vacía”, no está definido el valor que colocará en memoria. No está asociada esa variable con ningún contenido.

Si no vamos a utilizar las variables no debemos declararlas.

```
var num4 = 5;
```

JS

Se puede también declarar y asignar en la misma línea

*En los nombres de variables las mayúsculas y minúsculas **importan**.*

Var ≠ var

Constantes

Es similar al concepto de una variable, salvo que en este caso, la información que contiene es siempre la misma (**no puede variar** durante el flujo del programa).

A diferencia de las variables, comenzaremos con la palabra reservada **const** para indicarle al programa que es una constante. Su sintaxis es:

```
const CONSTANTE_1= value1;
```

```
const PI= 3.141592;  
const IVA= 21;
```

JS

En variables y constantes el caracter = (igual) actúa como caracter de asignación y se lee: “a la variable IVA le asignamos el valor 21”

Convención entre programadores: las constantes se colocan en MAYÚSCULAS y si los nombres de las constantes tienen más de una palabra se pueden poner un guion bajo, nunca dos palabras separadas. No se usa guion medio por convención.

Cambiando los valores...

Podemos cambiar el valor de una **variable** durante el flujo del programa...

```
var IVA= 21;  
IVA= 10.5;  
console.log(IVA);
```

JS

¿Qué valor tomará IVA?

```
const IVA= 21;  
IVA= 10.5;  
console.log(IVA);
```

JS

...no así con una **constante**.

¿Por qué sale este error?

```
✖ Uncaught TypeError: Assignment to constant variable.  
at ejemplo-JS.html:9
```


Tipos de datos

Las variables de JavaScript pueden contener muchos tipos de datos: numérico, cadena de caracteres, lógicos, indefinido, null, objetos y más.

El tipo de dato es la **naturaleza del contenido** de la variable o constante.

JavaScript tiene **tipado dinámico**, es decir que la misma variable se puede utilizar para contener diferentes tipos de datos:

```
var x;      // ahora x es indefinido (no tiene un valor definido)
x = 5;      // ahora es numérico (5)
x = "Juan"; // ahora es una cadena de caracteres o string ("Juan")
```

JS

JavaScript tiene un tipo numérico cuyos números se pueden escribir con o sin decimales:

```
var x1 = 34.00; // con decimales
var x2 = 34;    // sin decimales
```

JS

Tipado dinámico: JavaScript deduce cuál es el tipo de dato de la variable. El tipo de dato asociado a esa variable lo va a determinar el dato que estoy almacenando, por ejemplo: si almaceno un 5 se va a dar cuenta de que estoy almacenando un tipo de dato numérico, pero si almaceno "Juan" se va a dar cuenta de que almaceno un tipo de dato de tipo texto.

Tipos de datos

Los tipos de datos en JavaScript son los siguientes:

*Tipos de
datos
primitivos*

Tipo de dato	Descripción	Ejemplo básico
NUMBER number	Valor numérico (enteros, decimales, etc...)	42
STRING string	Valor de texto (cadenas de texto, caracteres, etc...)	'MZ'
BOOLEAN boolean	Valor booleano (valores verdadero o falso)	true
UNDEFINED undefined	Valor sin definir (variable sin inicializar)	undefined
FUNCTION function	Función (función guardada en una variable)	function() {}
OBJECT object	Objeto (estructura más compleja)	{}



Tipos de datos

El último estándar ECMAScript define nueve tipos:

- Seis tipos de datos primitivos
 - Undefined
 - Boolean
 - Number
 - String
 - BigInt
 - Symbol
- Null (tipo primitivo especial)
- Object
- Function

Para seguir investigando:

https://developer.mozilla.org/es/docs/Web/JavaScript/Data_structures



Identificando los tipos de datos

Para saber que tipo de dato tiene una variable, debemos observar que valor le hemos dado. Si es un valor numérico, será de tipo **number**. Si es un valor de texto, será de tipo **string**, si es verdadero o falso, será de tipo **booleano**:

```
var s = "Hola, me llamo Juan"; // s, de string
var n = 28; // n, de número
var b = true; // b, de booleano
var u; // u, de undefined
```

JS

Para saber qué tipo de dato tiene una variable utilizaremos **typeof()**, que solo sirve para variables con tipos de datos básicos o primitivos:

```
console.log(typeof s);
console.log(typeof n);
console.log(typeof b);
console.log(typeof u);
```

JS

string
number
boolean
undefined

Javascript determina los tipos de datos automáticamente. Muchas veces necesitaremos saber el tipo de dato de una variable e incluso convertirla a otros tipos de datos antes de usarla.

Los objetos

Son variables especiales que pueden contener más variables en su interior. Esto nos permite organizar múltiples variables de la misma temática dentro de un objeto.

```
const objeto = new Object(); // Esto es un objeto "genérico" vacío
```

JS

También podemos utilizar literales para declarar un objeto, que son las llaves `{ }`. Este ejemplo es equivalente al anterior, pero es más corto, rápido y cómodo, por lo que se aconseja su uso:

```
const objeto = {}; // Esto es un objeto vacío
```

JS

Este objeto tiene variables en su interior, a las cuales llamaremos **propiedades** que tendrán **valores**:

```
// Declaración del objeto
```

JS

```
const persona = {  
  nombre: "Juan",  
  apellido: "Perez",  
  edad: 35,  
};
```

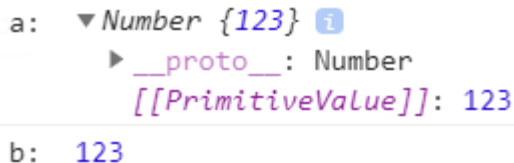
Objeto Number

Number es un objeto primitivo que permite representar y manipular valores numéricos. El constructor Number contiene constantes y métodos para trabajar con números. Valores de otro tipo pueden ser convertidos a números usando la función Number(). Su sintaxis es:

new Number(value)

```
var a = new Number('123');  
var b = Number('123'); // b es igual a 123  
console.log("a: ", a);  
console.log("b: ", b);
```

JS



a: ▼ Number {123} ⓘ
 ► __proto__: Number
 [[PrimitiveValue]]: 123
b: 123

*En el caso de **a** creamos el objeto a través de un constructor. Lo que se verá en la consola es el contenido del objeto.*

*En el caso de **b** se muestra el contenido de la variable.*

Ver ejemplo [number \(.html y.js\)](#)

Más info:

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Number

Las variables numéricas

En Javascript, los **números** son uno de los tipos de datos básicos (*tipos primitivos*) que para crearlos, simplemente basta con escribirlos. No obstante, en Javascript todo son objetos, como veremos más adelante, y también se pueden declarar como si fueran un objeto:

Constructor	Descripción
<code>NUMBER</code> <code>new Number(n)</code>	Crea un objeto numérico a partir del número <code>n</code> pasado por parámetro.
<code>NUMBER</code> <code>n</code>	Simplemente, el número en cuestión. Notación preferida.

Sin embargo, aunque existan varias formas de declararlos, no se suele utilizar la notación **new** con objetos primitivos ya que es bastante más tedioso y complicado que utilizar la notación de literales:

```
// Literales
const n1 = 4;
const n2 = 15.8;
// Objetos
const n1 = new Number(4);
const n2 = new Number(15.8);
```

JS

Comprobaciones numéricas

En Javascript tenemos varias funciones para conocer la naturaleza de una variable numérica (número finito, número entero, número seguro o si no es representable como un número). Las podemos ver a continuación en la siguiente tabla:

Método	Descripción
BOOLEAN <code>Number.isFinite(n)</code>	Comprueba si <code>n</code> es un número finito.
BOOLEAN <code>Number.isInteger(n)</code>	Comprueba si <code>n</code> es un número entero.
BOOLEAN <code>Number.isSafeInteger(n)</code>	Comprueba si <code>n</code> es un número seguro.
BOOLEAN <code>Number.isNaN(n)</code>	Comprueba si <code>n</code> no es un número.

Estas funciones devuelven un booleano (valor de *verdadero* o *falso*), lo que lo hace ideales para usarlas como condiciones en bucles o condicionales.

Comprobaciones numéricas

A continuación veamos dos ejemplos para cada una de estas funciones:

```
// ¿Número finito?  
Number.isFinite(42); // true  
Number.isFinite(Infinity); // false, es infinito  
// ¿Número entero?  
Number.isInteger(5); // true  
Number.isInteger(4.6); // false, es decimal  
// ¿Número seguro?  
Number.isSafeInteger(1e15); // true  
Number.isSafeInteger(1e16); // false, es un valor  
no seguro  
// ¿No es un número?  
Number.isNaN(NaN); // true  
Number.isNaN(5); // false, es un número
```

JS

Numero finito (42):	true
Numero finito (infinito):	false
Numero entero (5):	true
Numero entero (4.6):	false
Numero seguro (1e15):	true
Numero seguro (1e16):	false
Not a Number (NaN):	true
Not a Number (5):	false

[Ver ejemplo number \(.html y .js\)](#)

Conversión numérica

En muchos casos tendremos variables de texto que nos interesa convertir a número, para realizar operaciones posteriormente con ellas. Para ello, lo ideal es utilizar las funciones de parseo numérico, ***parseInt()*** y ***parseFloat()***. Veamos cuales son y cómo se pueden utilizar:

Método	Descripción
<small>NUMBER</small> Number.parseInt(<i>s</i>)	Convierte una cadena de texto <i>s</i> en un número entero.
<small>NUMBER</small> Number.parseInt(<i>s</i>, <i>radix</i>)	Idem al anterior, pero desde una base <i>radix</i> .
<small>NUMBER</small> Number.parseFloat(<i>s</i>)	Convierte una cadena de texto <i>s</i> en un número decimal.
<small>NUMBER</small> Number.parseFloat(<i>s</i>, <i>radix</i>)	Idem al anterior, pero desde una base <i>radix</i> .

Conversión numérica

Para ilustrar esto, veamos un ejemplo con **parseInt()** cuando solo le pasamos un parámetro (un texto) que queremos convertir a número:

```
Number.parseInt("42"); // 42  
Number.parseInt("42€"); // 42  
Number.parseInt("Núm. 42"); // NaN  
Number.parseInt("A"); // NaN
```

JS

parseInt (42)	42
parseInt (42\$)	42
parseInt (Num. 42)	NaN
parseInt (A)	NaN

[Ver ejemplo conversiones-numericas \(.html y.js\)](#)

La función **parseInt()** funciona perfectamente para variables de texto que contienen números o que empiezan por números. Esto es muy útil para eliminar unidades de variables de texto. Sin embargo, si la variable de texto comienza por un valor que no es numérico, **parseInt()** devolverá un **NaN (Not a Number)**.

Conversión numérica

Si lo que queremos es quedarnos con el número que aparece más adelante en la variable de texto, habrá que manipular ese texto con alguna de las funciones que veremos en el apartado de variables de texto.

Veamos ahora que ocurre si utilizamos ***parseInt()*** con dos parámetros, donde el primero es el texto con el número y el segundo es la base numérica del número:

```
Number.parseInt("11101", 2); // 29 en binario  
Number.parseInt("31", 8); // 25 en octal  
Number.parseInt("FF", 16); // 255 en hexadecimal
```

JS

*Ver ejemplo conversiones-
numericas (.html y.js)*

Esta modalidad de ***parseInt()*** se suele utilizar cuando queremos pasar a base decimal un número que se encuentra en otra base (binaria, octal, hexadecimal...).

parseInt (11101, 2 (binario))	29
parseInt (31, 8 (octal))	25
parseInt (FF, 16 (hexadecimal))	255

Al igual que con parseInt() tenemos otra función llamada parseFloat(). Funciona exactamente igual a la primera, sólo que la primera está específicamente diseñada para utilizar con números enteros y la segunda para números decimales. Si utilizamos parseInt() con un número decimal, nos quedaremos sólo con la parte entera, mientras que parseFloat() la conservará.

parseInt() y parseFloat()

Habíamos dicho que **parseInt()** convierte (parsea) un argumento de tipo cadena y devuelve un entero de la base especificada. Es una función de alto nivel y no está asociada a ningún objeto.

Sintaxis:

parseInt(string, base);

```
parseInt("F", 16); // F en hexadecimal = 15 en decimal
parseInt("15", 10); // devuelve 15 en decimal
parseInt("1111", 2); // 1111 en binario = 15 decimal
parseInt("15"); // devuelve 15 en decimal
```

JS

En cambio, **parseFloat()** convierte (parsea) un argumento de tipo cadena y devuelve un número de punto flotante. Sintaxis:

parseFloat(cadena)

```
parseFloat("3.14"); // devuelve el número 3.14
```

JS

Decimal	Binario	Hexadecimal
1	0000 0001	1
2	0000 0010	2
3	0000 0011	3
4	0000 0100	4
5	0000 0101	5
6	0000 0110	6
7	0000 0111	7
8	0000 1000	8
9	0000 1001	9
10	0000 1010	A
11	0000 1011	B
12	0000 1100	C
13	0000 1101	D
14	0000 1110	E
15	0000 1111	F
16	0001 0000	10

Objeto Math

Cuando trabajamos con Javascript, es posible realizar gran cantidad de operaciones matemáticas de forma nativa, sin necesidad de librerías externas.

Math es un objeto que tiene propiedades y métodos para constantes y funciones matemáticas. Todas las propiedades y métodos de Math son estáticos (no necesito llamar al constructor).

Constantes de Math

El objeto Math de Javascript incorpora varias constantes que podemos necesitar en algunas operaciones matemáticas:

Constante	Descripción	Valor
Math.E	Número de Euler e	2.718281828459045
Math.LN2	Logaritmo natural en base 2 $\ln 2$	0.6931471805599453
Math.LN10	Logaritmo decimal \log_{10}	2.302585092994046
Math.LOG2E	Logaritmo base 2 de E	1.4426950408889634
Math.LOG10E	Logaritmo base 10 de E	0.4342944819032518
Math.PI	Número PI π o Π	3.141592653589793
Math.SQRT1_2	Raíz cuadrada de 1/2	0.7071067811865476
Math.SQRT2	Raíz cuadrada de 2	1.4142135623730951

Math - Métodos matemáticos

Los siguientes métodos matemáticos están disponibles en Javascript a través del objeto **Math**. Observa que algunos de ellos sólo están disponibles en ECMAScript 6:

Método	Descripción	Ejemplo
<small>NUMBER</small> <code>Math.abs(x)</code>	Devuelve el <u>valor absoluto</u> $ W $ de x .	$ x $
<small>NUMBER</small> <code>Math.sign(x)</code> <small>ES2015</small>	Devuelve el signo del número: 1 positivo, -1 negativo	
<small>NUMBER</small> <code>Math.exp(x)</code>	<u>Exponenciación</u> W . Devuelve el número e elevado a x .	e^x
<small>NUMBER</small> <code>Math.expm1(x)</code> <small>ES2015</small>	Equivalente a <code>Math.exp(x) - 1</code> .	$e^x - 1$
<small>NUMBER</small> <code>Math.max(a, b, c...)</code>	Devuelve el número más grande de los indicados por parámetro.	
<small>NUMBER</small> <code>Math.min(a, b, c...)</code>	Devuelve el número más pequeño de los indicados por parámetro.	
<small>NUMBER</small> <code>Math.pow(base, exp)</code>	<u>Potenciación</u> W . Devuelve el número base elevado a exp .	$base^{exp}$
<small>NUMBER</small> <code>Math.sqrt(x)</code>	Devuelve la <u>raíz cuadrada</u> W de x .	\sqrt{x}
<small>NUMBER</small> <code>Math.cbrt(x)</code> <small>ES2015</small>	Devuelve la <u>raíz cúbica</u> W de x .	$\sqrt[3]{x}$
<small>NUMBER</small> <code>Math.imul(a, b)</code> <small>ES2015</small>	Equivalente a <code>a * b</code> , pero a nivel de bits.	
<small>NUMBER</small> <code>Math.clz32(x)</code> <small>ES2015</small>	Devuelve el número de ceros a la izquierda de x en binario (32 bits).	

Math - Métodos matemáticos

Veamos algunos ejemplos aplicados a las mencionadas funciones anteriormente:

```
Math.abs(-5); // 5
Math.sign(-5); // -1
Math.exp(1); // e, o sea, 2.718281828459045
Math.expm1(1); // 1.718281828459045
Math.max(1, 40, 5, 15); // 40
Math.min(5, 10, -2, 0); // -2
Math.pow(2, 10); // 1024
Math.sqrt(2); // 1.4142135623730951
Math.cbrt(2); // 1.2599210498948732
Math.imul(0xffffffff, 7); // -7

// Ejemplo de clz32 (count leading zeros)
const x = 1;
"0".repeat(Math.clz32(x)) + x.toString(2);
// Devuelve "0000000000000000000000000000000001"
```

JS

Abs:	5
Sign:	-1
Exp:	2.718281828459045
Expm1:	1.718281828459045
Max:	40
Min:	-2
Pow:	1024
Sqrt:	1.4142135623730951
Cbrt:	1.2599210498948732
Imul:	-7
clz32:	00000000000000000000000000000001

[Ver ejemplo math \(.html y.js\)](#)

Math - Método random()

Uno de los métodos más útiles e interesantes del objeto **Math** es **Math.random()**.

Método	Descripción	Ejemplo
<small>NUMBER</small> <code>Math.random()</code>	Devuelve un número al azar entre 0 y 1 con 16 decimales.	

Este método nos da un número al azar entre los valores **0** y **1**, con 16 decimales. Normalmente, cuando queremos trabajar con números aleatorios, lo que buscamos es obtener un número entero al azar entre **a** y **b**. Para ello, se suele hacer lo siguiente:

```
// Obtenemos un número al azar entre [0, 1) con 16 decimales
let x = Math.random();
// Multiplicamos dicho número por el valor máximo que buscamos (5)
x = x * 5;
// Redondeamos inferiormente, quedándonos sólo con la parte entera
x = Math.floor(x);
```

JS

Este ejemplo nos dará en x un valor al azar entre **0** y **5** (5 no incluido). Si presionamos F5 veremos el cambio en la consola.

0.3289342187867974

1.644671093933987

1

[Ver ejemplo random \(.html y.js\)](#)

Math - Métodos de redondeo

Es muy común necesitar métodos para redondear números y reducir el número de decimales o aproximar a una cifra concreta. Para ello, de forma nativa, Javascript proporciona los siguientes métodos de redondeo:

Método	Descripción
<small>NUMBER</small> <code>Math.round(x)</code>	Devuelve el redondeo de X (<i>el entero más cercano</i>)
<small>NUMBER</small> <code>Math.ceil(x)</code>	Devuelve el redondeo superior de X . (<i>el entero más alto</i>)
<small>NUMBER</small> <code>Math.floor(x)</code>	Devuelve el redondeo inferior de X . (<i>el entero más bajo</i>)
<small>NUMBER</small> <code>Math.fround(x)</code> <small>ES2015</small>	Devuelve el redondeo de X (<i>flotante con precisión simple</i>)
<small>NUMBER</small> <code>Math.trunc(x)</code> <small>ES2015</small>	Trunca el número X (<i>devuelve sólo la parte entera</i>)

Math - Métodos de redondeo

Veamos las diferencias de utilizar los diferentes métodos anteriores para redondear un número decimal y los resultados obtenidos:

```
// Redondeo natural, el más cercano
Math.round(3.75); // 4
Math.round(3.25); // 3
// Redondeo superior (el más alto)
Math.ceil(3.75); // 4
Math.ceil(3.25); // 4
// Redondeo inferior (el más bajo)
Math.floor(3.75); // 3
Math.floor(3.25); // 3
// Redondeo con precisión
Math.round(3.123456789); // 3
Math.fround(3.123456789); // 3.1234567165374756
// Truncado (sólo parte entera)
Math.trunc(3.75); // 3
Math.round(-3.75); // -4
Math.trunc(-3.75); // -3
```

JS

Round (natural):	4
Round (natural):	3
Ceil (superior):	4
Ceil (inferior):	4
Floor (inferior):	3
Floor (inferior):	3
Round (natural):	3
Fround (con precisión):	3.1234567165374756
Trunc (truncado):	3
Round (natural):	-4
Trunc (truncado negativo):	-3

*Ver ejemplo metodos-redondeo
(.html y.js)*

Math - Más métodos...

- **Referencia Developer Mozilla:**
https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Math
- **W3Schools:** https://www.w3schools.com/js/js_math.asp
- **Lenguaje JS:** <https://lenguajejs.com/javascript/fundamentos/objeto-math/>

Operadores aritméticos y de asignación

El **operador de asignación (=)** le otorga un valor a una variable y se coloca entre la variable y el valor a asignar.

```
var x = 10; JS
```

A la variable x le asignamos el valor 10

Los operadores aritméticos se utilizan para realizar operaciones aritméticas en números:

Operador	Descripción
+	Suma
-	Resta
*	Multiplicación
**	Exponenciación
/	División
%	Módulo: resto de dividir
++	Incremento
--	Decremento

Ver ejemplo operadores-aritmeticos (.html y.js)

Operadores de cadena y números

El operador + también se puede usar para agregar (concatenar) cadenas.

```
var txt1 = "Juan";  
var txt2 = "Pablo";  
var txt3 = txt1 + " " + txt2;  
console.log(txt3);
```

JS

Concatenamos ambas variables en una tercera y la mostramos por consola

Juan Pablo

El operador de asignación += también se puede usar para agregar (concatenar) cadenas:

```
var txt4 = "Bienvenidos ";  
txt4 += "a Javascript";  
console.log(txt4);
```

JS

A una variable con texto le concatenamos otro texto y la mostramos por pantalla

Bienvenidos a Javascript

Cuando se usa en cadenas, el operador + se denomina operador de concatenación.

Operadores de cadena y números

Agregar dos números devolverá la suma, pero agregar un número y una cadena devolverá una cadena:

```
var x = 5 + 5;  
var y = "5" + 5;  
var z = "Hola" + 5;
```

JS

Concatenamos ambas variables en una tercera y la mostramos por consola

10

55

Hola5

Ver ejemplo operadores-cadena (.html y.js)

La función prompt()

La función prompt es un **método** del objeto Window. Se utiliza para pedirle al usuario que ingrese datos por medio del teclado. Recibe dos parámetros: el mensaje que se muestra en la ventana y el valor inicial del área de texto. Su sintaxis es: *variable = prompt(mensaje, valor inicial)*

```
<script>  
  var nombre = prompt ("Ingrese su nombre", "")  
  document.write( "Hola " + nombre)  
</script>
```

JS

Ver ejemplo prompt.html

Ejemplos de JS y ejercicios

Ver ejemplo ejemplo-javascript (.html, .css y .js)

En este ejemplo se combina una característica interesante de JS que es la incorporar texto dentro del body con `document.write()`.

Además, se incorporan los métodos de *parseo* vista anteriormente, algunos estilos CSS y un Alert (Pop-Up).

Ver ejemplo ejemplo-javascript-2 (.html y .js)

Este ejemplo resume parte de los temas vistos hasta ahora, se debe inspeccionar el html y ver la consola. Probar agregando y quitando comentarios y actualizando la página.

Ejercicios

- Del archivo **“Actividad Práctica - JavaScript Unidad 1”** están en condiciones de hacer los ejercicios: 1 a 5
- **Ejercicio extra (combinando .js, .css y .html):** Crear una página que pida el nombre del usuario, dos valores y nos muestre las 4 operaciones aritméticas. Todos los datos deberán aparecer en el documento, incorporar estilos.

Video complementario - Introducción a JS:

<https://drive.google.com/file/d/10Yderet5Hk1VqppVPheNWv2UdeiYpYv0/view?usp=sharing>