

Contenidos a Trabajar

1. Introducción a la programación web con javascript
2. Historia de javascript
3. El estándar ECMAScript
4. Utilización de JavaScript
5. Conceptos sobre programación JS
6. Uso de consola
7. Uso inicial del lenguaje
8. Declaración de variables
9. Ámbito de variables
10. Tipos de uso básico de Datos
11. Conversión con Number, parseInt y parseFloat.

Introducción a la programación web con javascript

Javascript es un lenguaje capaz de aportar soluciones eficaces en la mayoría de los ámbitos de la tecnología.

Es especialmente importante porque es el único lenguaje de programación que entienden los navegadores, con el que se desarrolla la parte de la funcionalidad frontend en sitios web y aplicaciones web modernas. Pero también es fundamental en muchos otros tipos de desarrollos.

Sus usos más importantes son los siguientes:

- Desarrollo de sitios web del lado del cliente (frontend, en el navegador)
- Desarrollo de todo tipo de aplicaciones gracias plataformas del estilo NodeJS
- Desarrollo de aplicaciones para dispositivos móviles, híbridas o que compilan a nativo.
- Desarrollo de aplicaciones de escritorio para sistemas Windows, Linux y Mac, pudiendo escribir un código compatible con todas las plataformas.

Por tanto, podemos considerar a Javascript como un lenguaje de uso amplio y casi universal, pues abarca gran cantidad y tipos de aplicaciones y usos. Es por ello que resulta un lenguaje muy recomendable para aprender, ya que nos ofrece capacidades para usarlo en todo tipo de proyectos, siendo que algunos de ellos son casi exclusivos de Javascript.

Javascript es un lenguaje levemente tipado, que se presta bien para aprender a programar, ya que dar los primeros pasos es relativamente sencillo. Sin embargo, cuando el estudiante profundiza, sus características lo hacen diferente de otros lenguajes y requiere un estudio en profundidad para poder avanzar con garantías en cualquier aplicación avanzada.

Populares Librerías como jQuery o React, o frameworks como Angular, Vue o Ionic están basados en Javascript, sin olvidarnos de estándares ampliamente usados como Web Components y librerías basadas en ellos como Stencil o LitElement. Para abordar cualquiera de estas herramientas, y muchas otras, el aprendizaje sólido Javascript es un requisito importante.

Historia de Javascript

Javascript es un lenguaje creado por Netscape, la compañía propietaria de un navegador con el mismo nombre hoy desaparecido, que fue precursor del actual Firefox.

Su lanzamiento se produjo en 1995 en la versión de Netscape 2.0. Su autor, un programador llamado Brendan Eich, dice que lo creó en el tiempo de una semana.

Originalmente Javascript tomó el nombre de Mocha, aunque antes de su lanzamiento fue renombrado como LiveScript. Sin embargo, ese nombre se cambiaría finalmente a Javascript como consecuencia de un acuerdo entre Netscape y Sun Microsystems, que por aquella época era la propietaria del lenguaje Java. En virtud de ese acuerdo Netscape agregó compatibilidad para Java en su navegador, a la vez que tomaba el nombre de Javascript para su lenguaje. Esta denominación, sin embargo, ha provocado históricamente toda una serie de confusiones entre la comunidad, puesto que Javascript no tiene nada que ver con Java.

Para entender el enfoque de Javascript nos tenemos que trasladar a los primeros años de la web, en los que las páginas eran principalmente contenido y enlaces, puesto que por aquel entonces solo existía HTML. Sin embargo, era necesario aportar algún grado de interacción con el usuario, que fuera capaz de cubrir las necesidades básicas de los desarrolladores en aquella época. Así que se creó un lenguaje capaz de ejecutar pequeños programas en el contexto de una página web, con un juego de instrucciones capaz de interaccionar con

el usuario como respuesta a las acciones en la página. Pocos iban a suponer por entonces el futuro que le esperaba al lenguaje.

Entre tanto, la idea de un lenguaje liviano que se ejecuta en el navegador tuvo éxito y otros fabricantes la implementaron en sus clientes web. Es el caso de Microsoft, que bautizó a su lenguaje como JScript y fue presentado en 1996 con Internet Explorer 3. Javascript y JScript eran compatibles entre sí en un alto grado, pero igual que ocurría por aquel entonces con el lenguaje HTML, cada navegador hacía la guerra por su cuenta y construía su lenguaje innovando de la manera que le parecía oportuno.

Con el tiempo, las pequeñas diferencias entre Javascript y JScript se fueron haciendo más notorias y terminó representando un problema para desarrolladores y usuarios. Para ello en 1997 se produjo un movimiento para la estandarización del lenguaje, que acabó en la creación de ECMAScript, que no es más que el estándar del lenguaje Javascript.

Hoy afortunadamente, el Javascript que entienden todos los navegadores es el mismo, marcado por el estándar de ECMAScript. Dicho estándar no pudo tomar el nombre de Javascript, ya que éste es una marca comercial, propiedad actual de Oracle.

ECMAScript, el estándar

ECMAScript es el estándar creado para homogeneizar lenguaje Javascript implementado en cada navegador. El trabajo de estandarización de Javascript comenzó en 1997 y continúa evolucionando hasta la fecha.

ECMAScript realiza la especificación de Javascript, comenzando con la versión 1 hasta la versión 6, que es la más moderna en la implementación en navegadores. La versión ECMAScript 7, no obstante, ya se encuentra en proceso de especificación y algunas de sus funcionalidades ya están disponibles en algunos clientes web.

La organización que se encarga de esta estandarización es ECMA International, bajo pedido de Netscape. El nombre final del lenguaje fue acordado por los distintos fabricantes de navegadores, incluidos Netscape y Microsoft, junto con la propia ECMA.

Las versiones de ECMAScript se fueron sucediendo con rapidez los primeros años. ECMAScript 1 en 1997, ECMAScript 2 en el 98 y ECMAScript 3 en el 99. Sin embargo, luego hubo una brecha importante, incluso una versión abandonada (ECMAScript 4) porque sus propuestas no fueron apoyadas por toda la comunidad. La siguiente versión ECMAScript 5 se presentó en 2009 y aún hoy es la única que soportan todos los navegadores existentes en la actualidad (si se incluye Internet Explorer).

ECMAScript 5 supuso un avance importante con respecto a versiones anteriores. Incluyó el soporte para JSON, los getters y setters, el modo estricto y otra serie de mejoras, a la vez que se definió la implementación del modelo

de objetos en los navegadores (DOM, Document Object Model). Sin embargo, el lenguaje todavía tenía mucho potencial de mejora y las necesidades de las aplicaciones web dejaron notar que era necesaria una remodelación más profunda.

La especificación del estándar más revolucionario ha sido ECMAScript 6, también conocida como ES6 y cuyo nombre oficial es ECMAScript 2015, debido a su año de presentación. Esta versión mejoró de manera notable la orientación de objetos de Javascript, presentando clases y objetos en toda regla. También introdujo los módulos, toda una nueva gama de operadores, funciones lambda, iteradores, etc. ES6 está disponible hoy en todos los navegadores, aunque todavía algunos dispositivos u ordenadores viejos no lo puedan interpretar por disponer solamente de clientes web obsoletos.

A partir de ES6 se adquirió el compromiso de presentar nuevas especificaciones de ECMAScript cada año. Por lo que ES7 fue presentado en 2016, ES8 en 2017, ES9 en 2018 y así sucesivamente hasta hoy que vamos por la versión ES13 del 2022. Cada una de ella fue incorporando novedades y mejoras, aunque lo cierto es que los navegadores no las han ido implementando de manera generalizada.

Ingresando al mundo Javascript

Para entender bien lo que es Javascript y en qué situaciones se utiliza, debes conocer los distintos lenguajes básicos en la web. Aunque no es necesario que domines todos los lenguajes de la web para hacer un buen uso de Javascript, sí es importante que conozcas el HTML y CSS.

En la secuencia del aprendizaje de las tecnologías y lenguajes web, Javascript sería el lenguaje más adecuado para aprender después de conocer perfectamente HTML y CSS lo que hará que resulte sencillo incorporar posteriormente este lenguaje de programación. Dentro de sus características, en su nivel inicial es sencillo y pensado para hacer las cosas con rapidez, a veces con ligereza. Incluso las personas que no tengan una experiencia previa en la programación podrán aprender este lenguaje con facilidad y utilizarlo en toda su potencia con sólo un poco de práctica.

Cuando comiences el aprendizaje de Javascript, ejecutando programas en el navegador, observarás la creación de interfaces de usuario dinámicas, que permitirán personalizar la experiencia de uso de una página web, con elementos que respondan a las acciones del usuario de una manera avanzada y personalizada. Aunque también Javascript nos permite ejecutar instrucciones como respuesta a las acciones del usuario (eventos), con lo que podemos crear páginas interactivas y aplicaciones. Este es otro de los grandes focos de la web, incorporar poco a poco una gran cantidad de utilidades para uso personal y profesional, con programas sencillos como una calculadoras, una agendas, hasta programas tan complejos como un software de gestión empresarial que es capaz de comportarse en la web de manera similar a una aplicación de escritorio.

Para acceder a toda esta gama de funcionalidades, Javascript pone a disposición del programador todos los elementos que forman la página web, para programarlos y manipularlos dinámicamente. Con Javascript el programador es capaz de alterar cualquier cosa que se muestra en una página, cambiando, insertando o eliminando todo tipo de contenido. Podés controlar cada cosa que ocurre en la página cuando la está visualizando el usuario y comunicar con él y con todo tipo de interfaces especiales. Todo eso es lo que permitirá crear aplicaciones web realmente impactantes.

Conceptos sobre programación en Javascript

Javascript, el lenguaje del lado del cliente

A Javascript se le denomina lenguaje "del lado del cliente" porque se ejecuta en contexto del navegador (cliente web), en contraposición a lenguajes como PHP, que se ejecutan del "lado del servidor". En el lado del cliente es el navegador el que soporta la carga de procesamiento y también es el que nos aporta los recursos con los que contamos para programar las aplicaciones.

Desde hace años, gracias a su compatibilidad con todos los navegadores existentes en el mercado, se ha convertido en un estándar para la programación del lado del cliente. Con Javascript podemos crear efectos de animación gráfica y hacer interactivas las páginas web en relación con el usuario. El navegador del cliente es el encargado de interpretar las instrucciones Javascript y ejecutarlas para realizar estos efectos e interacción, de modo que el mayor recurso con que cuenta este lenguaje es el propio navegador y todos los elementos que hay dentro de una página (que no es poco).

De manera adicional, gracias a las API Javascript de HTML5, que están disponibles en los navegadores actuales de ordenadores y dispositivos, puede acceder a recursos adicionales, como la cámara web, espacio para almacenamiento de datos, creación de gráficos basados en vectores y mapas

de bits, flujos de datos con servidores, etc. Todo ello ha multiplicado las posibilidades del lenguaje.

Entonces, aunque Javascript es conocido principalmente como un lenguaje del lado del cliente, lo cierto es que Javascript es mucho más. Hoy es posible ampliar el contexto de Javascript a muchos ámbitos y para iniciarse con este lenguaje, lo ideal es inicialmente utilizarlo dentro del navegador.

Javascript como lenguaje integrador

Aunque Javascript surgió con el objetivo inicial de programar ciertos comportamientos sobre las páginas web, respondiendo a la interacción del usuario y la realización de acciones sencillas, lo cierto es que hoy sus ámbitos de trabajo son mucho mayores. Ha dejado de ser un "lenguaje de scripting" del lado del cliente, para convertirse en un lenguaje integrador de múltiples ámbitos y prestaciones.

Encontramos Javascript, ya no solo en la Web, también es nativo en sistemas operativos como Windows. También es capaz de hacer programas de consola, bajo plataformas como NodeJS, así como programas de escritorio multiplataforma (Windows, Linux y Mac). Paralelamente, es posible usar Javascript para el desarrollo de aplicaciones en dispositivos (apps híbridas) y aplicaciones que se compilan a código nativo de celulares.

En resumen, ver Javascript como el lenguaje utilizado para crear pequeños programas encargados de realizar acciones dentro del ámbito de una página web ha quedado muy lejos de la realidad actual. Hoy aplicaciones web muy complejas y de alto tránsito están desarrolladas en gran medida con

Javascript. Desarrollos inmensos como Gmail, Netflix, Facebook, Twitter, Outlook, editores de código como Atom o VS code y aplicaciones de mensajería como Slack o terminales como Hyper y gran número de Apps para móviles, o juegos exitosos como Candy Crush han sido desarrollados con este lenguaje.

Uso de Consola en Javascript

Todos los navegadores web modernos incluyen una consola, una herramienta creada para desarrolladores, usuarios avanzados o cualquier persona.

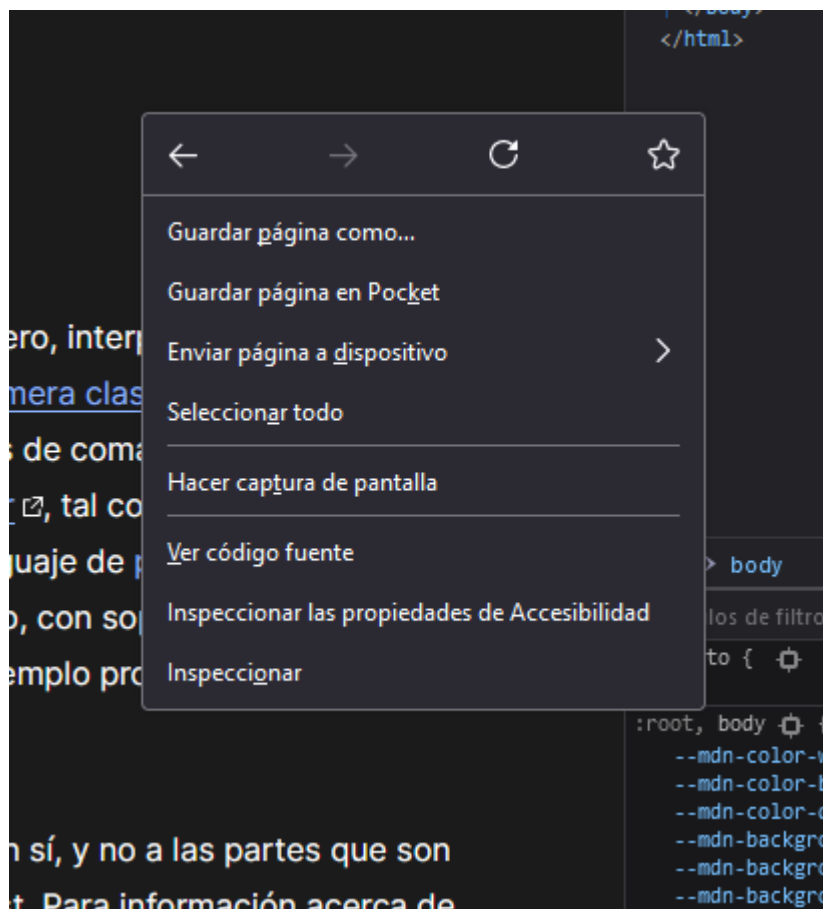
Tiene como finalidad mostrar mensajes de información, error o alerta que se reciben al hacer las peticiones para cargar desde la red los elementos incluidos en las páginas. Además incluye inspectores y verdaderos depuradores de código. También permite interactuar con la página, ejecutando expresiones o comandos de JavaScript.

El propósito es probar el funcionamiento de las páginas o aplicaciones y descubrir posibles errores en el código.

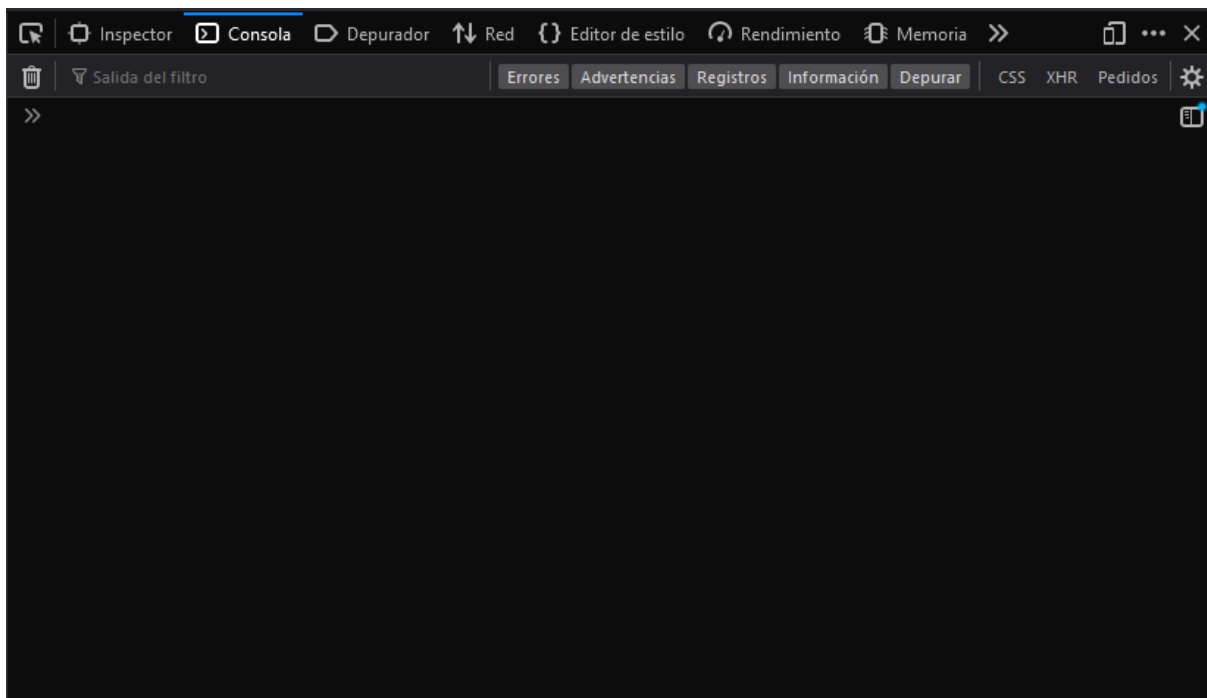
Esta consola la utilizan los desarrolladores, pero cualquier usuario puede probarla y así estar al tanto del contenido de las páginas de cualquier sitio web. La consola de JavaScript es un panel que se abre en alguna porción pequeña del navegador para no interferir con el contenido principal, aunque se puede mover y cambiar su tamaño. Cada navegador web posee un aspecto y se abre de forma diferente.

¿Cómo acceder a las herramientas de desarrollador?

La forma más sencilla es dar click derecho sobre la pantalla del navegador y seleccionar la opción “inspeccionar”.



Otra alternativa es presionar la tecla f12 o el conjunto de teclas ctrl + Mayús + J.



Una vez listo veremos que se despliega una ventana dentro del mismo navegador con una lista de pestañas de diferentes opciones en las que se puede encontrar información de las peticiones HTTP (los elementos de la página que se cargan de la red), los errores y análisis de CSS, manejo de JavaScript por consola, errores y advertencias de seguridad y los Logs (mensajes).

Cómo crear mensajes para mostrar en la consola del navegador

La consola se puede emplear en conjunto con JavaScript en muchas operaciones y para mostrar un sencillo mensaje en una página web, solo es necesario incluir en cualquier parte de la página o de un archivo HTML el siguiente script:

```
> console.log("Texto del mensaje");  
Texto del mensaje
```

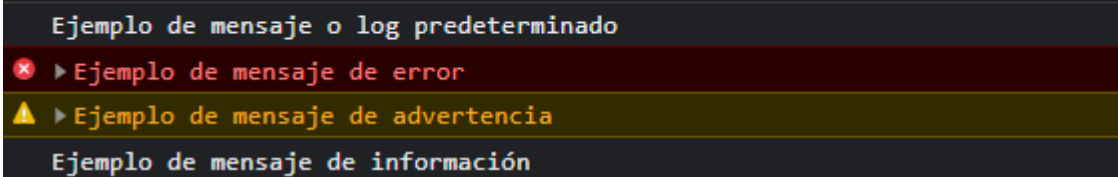
Como podrás observar, el código JavaScript posee las etiquetas `<script></script>` para indicarle al navegador que el código existente en esas etiquetas debe ser procesado como lenguaje javascript. En este caso "console.log()" con un mensaje de texto dentro de los paréntesis, indican el mensaje a mostrar en la consola.

Métodos para mostrar mensajes en la consola del navegador

- El método anterior "console.log()" muestra solo un mensaje, sin clasificarlo.
- El método "console.info()" muestra un icono azul de información y a continuación el texto del mensaje. En algunos exploradores, se muestra simplemente como el .log().
- El método "console.error()" muestra un icono rojo de error con el mensaje coloreado en rojo.
- El método "console.warn()" muestra un icono amarillo de advertencia.

Por ejemplo podés utilizar el siguiente código y ver el resultado del mismo en la consola de tu navegador escribiendo el siguiente script:


```
> console.log("Ejemplo de mensaje o log predeterminado");  
console.error("Ejemplo de mensaje de error");  
console.warn("Ejemplo de mensaje de advertencia");  
console.info("Ejemplo de mensaje de información");
```



Otros métodos de la consola

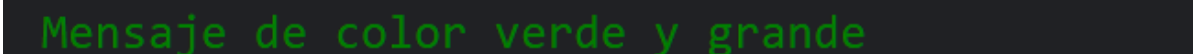
- `console.dir(object)`. Imprime una representación del objeto con JavaScript. Por ejemplo: "`console.dir(document.links);`" muestra todos los enlaces en la página.
- `console.dirxml(object)`. Imprime una representación en XML de los elementos descendientes del objeto.
- `console.clear()`. Limpia la ventana de la consola.

Personalizar el formato del mensaje en la consola del navegador

Es posible aplicar formato al mensaje de texto usando CSS, aunque algunas configuraciones solo se verán correctamente en los navegadores Google Chrome y Edge.

Por ejemplo, para darle el color verde y usar una fuente grande usa:

```
> console.log("%cMensaje de color verde y grande", "color: green; font-size: x-large");
```



Crear tablas en la consola del navegador

En la consola se pueden mostrar tablas, usando el método "console.table()".

No es necesario preocuparse por el formato, solo especificar los datos.

A continuación algunos ejemplos.

Comprabá el resultado en la consola.

Tabla simple con dos columnas - objeto.

```
> console.table({ Nombre : "Pepe", Apellido : "Grillo" });
```

(index)	Value
Nombre	'Pepe'
Apellido	'Grillo'
► Object	

Tabla con tres columnas - array.

```
> var gente = [ ["Norfi", "Carrodegas"], ["Pedro", "Perez"], ["Juan", "Lopez"] ];  
console.table(gente);
```

(index)	0	1
0	'Norfi'	'Carrodegas'
1	'Pedro'	'Perez'
2	'Juan'	'Lopez'
► Array(3)		

Convenciones y uso inicial del lenguaje.

El lenguaje Javascript tiene una sintaxis muy parecida a Java y a lenguaje C, de modo que si el conocés alguno de estos dos lenguajes podrás manejarte con facilidad iniciando este código. De todos modos, vamos a describir la sintaxis inicial, ya que es siempre conveniente tener esa información.

Comentarios en el código

Un comentario es una parte del código que no es interpretada por el navegador y cuya utilidad radica en facilitar la lectura al programador. El programador, a medida que desarrolla el script, va dejando frases o palabras sueltas, llamadas comentarios, que le ayudan a él o a cualquier otro a leer más fácilmente el script a la hora de modificarlo o depurarlo.

Existen dos tipos de comentarios en el lenguaje. Uno de ellos, la doble barra, sirve para comentar una línea de código. El otro comentario lo podemos utilizar para comentar varias líneas y se indica con los signos `/*` para empezar el comentario y `*/` para terminarlo.

Veamos unos ejemplos:

```
//Este es un comentario de una línea  
  
/**  
 * Este comentario se puede  
 * extender por varias líneas.  
 * Las que quieras.-  
 */
```

Mayúsculas y minúsculas

En Javascript se han de respetar las mayúsculas y las minúsculas. Si nos equivocamos al utilizarlas el navegador responderá con un mensaje de error, ya sea de sintaxis o de referencia indefinida. Por ejemplo, no es lo mismo la función `alert()` que la función `Alert()`. La primera muestra un texto en una caja de diálogo y la segunda (con la primera A mayúscula) simplemente no existe, a no ser que la definamos nosotros.

Entonces, para que la función no arroje errores de Javascript, se tiene que escribir toda en minúscula. Otro claro ejemplo lo veremos cuando tratemos con variables, puesto que los nombres que damos a las variables también son sensibles a las mayúsculas y minúsculas.

Por regla general, los nombres de las cosas en Javascript se escriben siempre en minúsculas, salvo que se utilice un nombre con más de una palabra, pues en ese caso se escribirán con mayúsculas las iniciales de las palabras siguientes a la primera.

Por ejemplo `document.bgColor` (que es un lugar donde se guarda el color de fondo de la página web), se escribe con la "C" de color en mayúscula, por ser la primera letra de la segunda palabra. También se puede utilizar mayúsculas en las iniciales de las primeras palabras en algunos casos, como los nombres de las clases, veremos más adelante cuáles son estos casos y qué son las clases.

Separación de instrucciones

Las distintas instrucciones que contienen nuestros scripts se han de separar convenientemente para que el navegador no indique los correspondientes errores de sintaxis. Javascript tiene dos maneras de separar instrucciones. La primera es a través del carácter punto y coma (;) y la segunda es a través de un salto de línea.

Por esta razón Las sentencias Javascript no necesitan acabar en punto y coma a no ser que coloquemos dos instrucciones en la misma línea, sin embargo es considerado una buena práctica conservar el uso del punto y coma al finalizar nuestras declaraciones de Javascript.

Variables en Javascript

Concepto de variable

Una variable es un espacio en memoria donde se almacena un dato, un espacio donde podemos guardar cualquier tipo de información que necesitemos para realizar las acciones de nuestros programas. Podemos pensar en ella como una caja, donde almacenamos un dato. Esa caja tiene un nombre, para que más adelante podamos referirnos a la variable, recuperar el dato así como asignar un valor a la variable siempre que deseemos.

Variables en Javascript

Por ejemplo, si nuestro programa realiza sumas, será muy normal que guardemos en variables los distintos valores que participan en la operación y el resultado de la suma. El efecto sería algo parecido a esto.

```
var sumando1 = 23;  
var sumando2 = 33;  
var sumatotal= sumando1 + sumando2;
```

En este ejemplo tenemos tres variables, sumando1, sumando2 y sumatotal, donde guardamos el resultado. Vemos que su uso para nosotros es como si tuviésemos un apartado donde guardar un dato y que se pueda acceder a ellos con sólo poner su nombre.

Reglas para el nombrado de variables en Javascript

Los nombres de las variables han de construirse con caracteres alfanuméricos (números y letras), el carácter subrayado o guión bajo (_) y el carácter dólar \$. Además de esta regla, hay una serie de reglas adicionales para construir nombres para variables.

Lo más importante es que no pueden comenzar por un carácter numérico. No podemos utilizar caracteres raros como el signo +, un espacio o un signo -. Nombres admitidos para las variables podrían ser:

Edad

paisDeNacimiento

_nombre

\$elemento

Otro\$_Nombres

También hay que evitar utilizar nombres reservados como variables, por ejemplo no podremos llamar a nuestra variable palabras como return o for, que ya veremos que son utilizadas para estructuras del propio lenguaje.

Veamos ahora algunos nombres de variables que no está permitido utilizar:

12meses

tu nombre

```
return  
for  
mas-o-menos  
pe%pe
```

Los nombres de variables en Javascript son sensibles a mayúsculas y minúsculas. Recordá que Javascript es un lenguaje sensible a mayúsculas y minúsculas, por lo que las variables también se afectan por esa distinción. Por lo tanto, no es lo mismo la variable de nombre "minombre" que la variable "miNombre". No es lo mismo "Edad" que "edad".

Tené muy en cuenta este detalle, ya que es una habitual fuente de problemas aprendiendo a desarrollar utilizando este lenguaje y que en el código generar errores difíciles de detectar. Esto es porque estás usando una variable, que debería tener un dato determinado, pero si te equivocás al escribirla y usas mayúsculas o minúsculas donde no debería, entonces será otra variable diferente, que no tendrá el dato que se espera. Como Javascript no te obliga a declarar las variables el programa se ejecutará sin producir un error, sin embargo, en la ejecución no producirá los efectos deseados.

Declaración de variables en Javascript

Declarar variables consiste en definir, y de paso informar al sistema, que vas a utilizar una variable. Es una costumbre habitual en los lenguajes de programación el especificar explícitamente las variables que se van a usar en los programas. En muchos lenguajes de programación hay unas reglas estrictas a la hora de declarar las variables, pero lo cierto es que Javascript es bastante permisivo.

Javascript se salta muchas reglas por ser un lenguaje un tanto libre a la hora de programar y uno de los casos en los que otorga un poco de libertad es a la hora de declarar las variables, ya que no estamos obligados a hacerlo, al contrario de lo que pasa en otros lenguajes de programación como Java, C, C# y otros. La declaración de variables con var Javascript cuenta con la palabra "var" que utilizaremos cuando queramos declarar una o varias variables. Como es lógico, se utiliza esa palabra para definir la variable antes de utilizarla.

Nota: Aunque Javascript no nos obliga a declarar explícitamente las variables, es aconsejable declararlas antes de utilizarlas y veremos en adelante que se trata también de una buena costumbre.

Además, veremos que en algunos casos especiales, no producirá exactamente los mismos resultados un script en el que hemos declarado una variable y otro en el que no lo hayamos hecho, ya que la declaración o no afecta al ámbito de las variables.

```
var operando1;  
var operando2;
```

También se puede asignar un valor a la variable cuando se está declarando

```
var operando1 = 23;  
var operando2 = 33;
```

También se permite declarar varias variables en la misma línea, siempre que se separen por comas.

```
var operando1,operando2;
```

Declaración de variables Javascript con let y const

Javascript es un estándar y como tal va evolucionando en el tiempo, en versiones modernas y en concreto en Javascript en su versión ES6, existen otros modos de declarar variables:

Declaración **let**: Esta nueva manera de declarar las variables afecta a su ámbito, ya que son locales al bloque donde se están declarando.

Declaración **const**: En realidad "const" no declara una variable sino una constante, que no puede variar su valor a lo largo de la ejecución de un programa.

Ámbito de las variables en Javascript

El ámbito de las variables es uno de los conceptos más importantes que deberemos conocer cuando trabajamos con variables, no sólo en Javascript, sino en la mayoría de los lenguajes de programación.

Es de destacar los cambios al ámbito de las variables declaradas con "var" y de la forma más reciente de declarar variables, con "let", que también afecta directamente a su ámbito.

Concepto de ámbito de variables

Se le llama ámbito de las variables al lugar donde estas están disponibles. Por lo general, cuando declaramos una variable hacemos que esté disponible en el lugar donde se ha declarado, esto ocurre en todos los lenguajes de programación y como Javascript se define dentro de una página web, las variables que declaremos en la página estarán accesibles dentro de ella.

En Javascript no podremos acceder a variables que hayan sido definidas en otra página. Por tanto, la propia página donde se define es el ámbito más habitual de una variable y le llamaremos a este tipo de variables globales a la página. También se pueden hacer variables con ámbitos distintos del global, es decir, variables que declaremos y tendrán validez en lugares más acotados.

Variables globales

Son variables globales las que están declaradas en el ámbito más amplio posible, que en Javascript es una página web. Para declarar una variable global a la página simplemente lo haremos en un script, con la palabra var.

```
var variableGlobal;
```

Las variables globales son accesibles desde cualquier lugar de la página, es decir, desde el script donde se han declarado y todos los demás scripts de la página, incluidos los manejadores de eventos, los que veremos más adelante.

Variables locales

También podremos declarar variables en lugares más cerrados, como por ejemplo una función. A estas variables las llamaremos locales. Cuando se declaren variables locales sólo podremos acceder a ellas dentro del lugar donde se ha declarado, es decir, si la habíamos declarado en una función solo podremos acceder a ella cuando estemos en esa función.

Las variables pueden ser locales a una función, pero también pueden ser locales a otros ámbitos, como por ejemplo un bucle. En general, son ámbitos locales cualquier lugar acotado por llaves.

```
function miFuncion (){  
    var variableLocal;  
}
```

En el script anterior hemos declarado una variable dentro de una función, por lo que esa variable sólo tendrá validez dentro de la función. Se pueden ver cómo se utilizan las llaves para acotar el lugar donde está definida esa función o su ámbito.

No hay problema en declarar una variable local con el mismo nombre que una global, en este caso la variable global será visible desde toda la página, excepto en el ámbito donde está declarada la variable local ya que en este sitio ese nombre de variable está ocupado por la local y es ella quien tiene validez. En resumen, la variable que tendrá validez en cualquier sitio de la página es la global.

Menos en el ámbito donde está declarada la variable local, que será ella quien tenga validez.

```
var numero = 2;  
  
function miFuncion (){  
    var numero = 19;  
    document.write(numero); //imprime 19  
}  
  
document.write(numero); //imprime 2
```

Es una buena práctica no declarar variables con los mismos nombres, para que no haya lugar a confusión sobre qué variable es la que tiene validez en cada momento.

Ámbito de variables declaradas con let y const

let y const son dos formas de declarar variables en JavaScript introducidas en ES6 que reducen el ámbito de la variable a bloques (con var el ámbito era la función actual) y no admiten hoisting. Además, las variables declaradas con const no pueden ser reasignadas (aunque no significa que su valor sea inmutable, como veremos a continuación).

let

Un bloque en JavaScript se puede entender como «lo que queda entre dos corchetes», ya sean definiciones de funciones o bloques if, while, for y loops similares. Si una variable es declarada con let en el ámbito global o en el de una función, la variable pertenecerá al ámbito global o al ámbito de la función respectivamente, de forma similar a como ocurría con var.

Por ejemplo, en el siguiente snippet la variable i es una variable global y la variable j es una variable local:

```
let i = 0;
function foo() {
  i = 1;
  let j = 2;
  if(true) {
    console.log(i); // 1
    console.log(j); // 2
  }
}
foo();
```

Pero si declaramos una variable con `let` dentro un bloque que a su vez está dentro de una función, la variable pertenece solo a ese bloque:

```
function foo() {
  let i = 0;
  if(true) {
    let i = 1; // Sería otra variable i solo para el bloque if
    console.log(i); // 1
  }
  console.log(i); // 0
}
foo();
```

Fuera del bloque donde se declara con `let`, la variable no está definida:

```
function foo() {  
  if(true) {  
    let i = 1;  
  }  
  console.log(i); // ReferenceError: i is not defined  
}  
foo();
```

Debido a este comportamiento, muchos desarrolladores se inclinan hacia `let` como la forma predeterminada de declarar variables en JavaScript y abandonar `var` (1, 2, 3), pues el scope más específico previene la sobreescritura de variables de forma accidental al declarar variables sin ensuciar el scope superior.

const

El ámbito o scope para una variable declarada con `const` es, al igual que con `let`, el bloque, pero si la declaración con `let` previene la sobreescritura de variables, `const` directamente prohíbe la reasignación de valores (`const` viene de `constant`).

Si se intenta reasignar una variable constante se obtendrá un error tipo `TypeError`:

```
const i = 0;  
i = 1; // TypeError: Assignment to constant variable
```


Hoisting

Además del ámbito de aplicación visto antes, una variable declarada con `var` es sometida a hoisting («izamiento» o «levantamiento»): la declaración de la variable es «levantada» hasta el inicio del ámbito de aplicación pero la asignación al valor permanece en el sitio donde se realice.

Si intentamos acceder a su valor antes de que se asigne el valor, obtendremos un valor indefinido (`undefined`):

```
console.log(i); // undefined
var i = 1;
```

Este comportamiento se puede entender como «la variable `i` ha sido declarada en el programa pero en el momento de intentar acceder a ella aún no tenía un valor asignado». La interpretación sería similar al siguiente código:

```
var i; // Variable declarada pero valor no definido
console.log(i); // undefined
i = 1;
console.log(i); // 1
```

Sin embargo, si la variable no es declarada en absoluto obtendremos un `ReferenceError`, que no es lo mismo que obtener un valor indefinido:

```
console.log(x); // ReferenceError: x is not defined  
var i = 1;
```

Debido a este comportamiento, se suele recomendar mover todas las declaraciones de variables al inicio del scope aunque no se asigne valor alguno, de esta forma se evitan estos posibles errores de ejecución.

El hoisting no es posible en variables declaradas con `let` o `const`. Estas variables siempre darán un `ReferenceError` si se intenta acceder a ellas antes de que sean declaradas:

```
console.log(x); // ReferenceError: x is not defined  
let x = 1;
```

Tipos de Datos

En una variable podemos introducir varios tipos de información. Por ejemplo podríamos introducir texto simple, números enteros o reales, etc. A estas distintas clases de información se les conoce como tipos de datos. Cada uno tiene características y usos distintos.

Tipos de datos de uso habitual en Javascript.

Number

Para empezar tenemos el tipo Number, para guardar números como 9 o 23.6.

En este lenguaje sólo existe un tipo de datos numérico, al contrario que ocurre en la mayoría de los lenguajes más conocidos. Todos los números son por tanto del tipo numérico, independientemente de la precisión que tengan o si son números reales o enteros. Los números enteros son números que no tienen coma, como 3 o 339. Los números reales son números fraccionarios, como 2.69 o 0.25, que también se pueden escribir en notación científica, por ejemplo 2.482e12.

Con Javascript también podemos escribir números en otras bases, como la hexadecimal. Las bases son sistemas de numeración que utilizan más o menos dígitos para escribir los números. Existen tres bases con las que podemos trabajar Base 10, es el sistema que utilizamos habitualmente, el sistema

decimal. Cualquier número, por defecto, se entiende que está escrito en base 10.

Base 8, también llamado sistema octal, que utiliza dígitos del 0 al 7. Para escribir un número en octal basta con escribir ese número precedido de un 0, por ejemplo 045.

Base 16 o sistema hexadecimal, es el sistema de numeración que utiliza 16 dígitos, los comprendidos entre el 0 y el 9 y las letras de la A a la F, para los dígitos que faltan. Para escribir un número en hexadecimal debemos escribirlo precedido de un cero y una equis, por ejemplo 0x3EF.

String

El tipo String guarda un texto. Siempre que escribamos una cadena de caracteres debemos utilizar las comillas dobles (""), simples (") o invertidas (``).

Javascript sólo tiene un tipo de datos para guardar texto y en el se pueden introducir cualquier número de caracteres. Un texto puede estar compuesto de números, letras y cualquier otro tipo de caracteres y signos. Los textos se escriben entre comillas, dobles o simples.

```
miTexto = "Pepe se va a pescar";  
miTexto = '23%%$ Letras & *--*';
```

Todo lo que se coloca entre comillas, como en los ejemplos anteriores, es tratado como una cadena de caracteres independientemente de lo que coloquemos en el interior de las comillas. Por ejemplo, en una variable de texto podemos guardar números y en ese caso tenemos que tener en cuenta que las variables de tipo texto y las numéricas no son la misma cosa y mientras que las de numéricas nos sirven para hacer cálculos matemáticos las de texto no.

Caracteres de escape en cadenas de texto.

Hay una serie de caracteres especiales que sirven para expresar en una cadena de texto determinados controles como puede ser un salto de línea o un tabulador. Estos son los caracteres de escape y se escriben con una notación especial que comienza por una contra barra (una barra inclinada al revés de la normal `"`) y luego se coloca el código del carácter a mostrar.

Un carácter muy común es el salto de línea, que se consigue escribiendo `\n`. Otro carácter muy habitual es colocar unas comillas, pues si colocamos unas comillas sin su carácter especial nos cerrarían las comillas que colocamos para iniciar la cadena de caracteres. Las comillas las tenemos que introducir entonces con `\"` o `\'` (comillas dobles o simples). Existen otros caracteres de escape, que veremos en la tabla de abajo más resumidos, aunque también hay que destacar como carácter habitual el que se utiliza para escribir una contrabarra, para no confundirla con el inicio de un carácter de escape, que es la doble contrabarra.

Tabla con todos los caracteres de escape:

Salto de línea: `\n`

Comilla simple: \'

Comilla doble: \"

Tabulador: \t

Retorno de carro: \r

Avance de página: \f

Retroceder espacio: \b

Contrabarra: \\

Boolean

También contamos con el tipo Boolean, que guarda una información que puede valer si (true) o no (false).

El tipo booleano, boolean en inglés, sirve para guardar un sí o un no o dicho de otro modo, un verdadero o un falso. Se utiliza para realizar operaciones lógicas, generalmente para realizar acciones si el contenido de una variable es verdadero o falso.

Si una variable es verdadera entonces --> Ejecuto unas instrucciones.

Si no --> Ejecuto otras.

Los dos valores que pueden tener las variables booleanas son true o false.

```
esVerdadero = true;  
esFalso = false;
```

Por último sería relevante señalar aquí que nuestras variables pueden contener cosas más complicadas, como podría ser un objeto, una función, o vacío (null) lo que veremos más adelante.

En realidad las variables no están forzadas a guardar un tipo de datos en concreto y por lo tanto no especificamos ningún tipo de datos para una variable cuando la estamos declarando. Podemos introducir cualquier información en una variable de cualquier tipo, incluso podemos ir cambiando el contenido de una variable de un tipo a otro sin ningún problema.

Vamos a ver esto con un ejemplo:

```
var nombre_ciudad = "Valencia";  
var revisado = true;  
nombre_ciudad = 32;  
revisado = "no";
```

Esta ligereza a la hora de asignar tipos a las variables puede ser una ventaja en un principio, sobretudo para personas inexpertas, pero a la larga puede ser fuente de errores ya que dependiendo del tipo que son las variables se comportarán de un modo u otro y si no controlamos con exactitud el tipo de las variables podemos encontrarnos sumando un texto a un número. Javascript operará perfectamente, y devolverá un dato, pero en algunos casos puede que no sea lo que estábamos esperando. Así pues, aunque tenemos libertad con los tipos, esta misma libertad nos hace estar más atentos a posibles desajustes difíciles de detectar a lo largo de los programas.

Veamos lo que ocurriría en caso de sumar letras y números.

```
var sumando1 = 23;  
var sumando2 = "33";  
var suma = sumando1 + sumando2;  
console.log(suma); // 2333
```

Este script nos mostraría en la página el texto 2333, que no se corresponde con la suma de los dos números, sino con su concatenación, uno detrás del otro.

Funciones parseInt(), parseFloat(), Number() y conversión implícita.

Veremos las posibilidades que ofrece el lenguaje para convertir otros tipos en números.

Detallaremos las funciones parseInt(), parseFloat(), Number() y las formas de conversión rápida (conversión implícita), comentando las diferencias entre ellas.

parseInt() y parseFloat()

Son funciones creadas para analizar un string y devolver un número si es posible. Los espacios iniciales y finales se ignoran.

JavaScript analiza la cadena para extraer las cifras que encuentre al principio. Estas cifras al principio del string son las que se transforman a tipo numérico. Cuando se encuentra el primer carácter no numérico se ignora el resto de la cadena. Si el primer carácter encontrado no es convertible a número, el resultado será NaN (Not a Number).

Cuando el valor no es un string, javascript hace primero una conversión implícita a string. Esta conversión, en el caso de objetos, se hace llamando al método toString(). Podemos reescribir este método para algunos objetos si nos interesa asegurarnos de que devuelvan un string convertible a número.

parseInt(string, radix) - Conversión de string a entero

radix es opcional y representa la base en la que estamos trabajando. Normalmente trabajaremos en base decimal y este será el valor tomado por defecto en los navegadores modernos (como se define desde ECMAScript 5).

El prefijo 0x indica que el número está en hexadecimal aunque no se incluya radix 16.

Algunos ejemplos:

```
parseInt("10"); // 10
parseInt("10.8"); // 10
parseInt("10 22"); // 10
parseInt(" 14 "); // 14
parseInt("20 dias"); // 20
parseInt("Hace 20 dias"); // NaN
parseInt("44aa33bb"); // 44
parseInt("3.14"); // 3
parseInt("314e-2"); // 314
parseInt(""); // NaN -> ¡¡el string vacío se convierte a NaN!!
parseInt(null); // NaN
parseInt("10"); // 10
parseInt("010"); // 10
parseInt("10"); // 8
parseInt("0x10"); // 16 0x indica que el número es hexadecimal
parseInt("10",16); //16
```

parseFloat(string) - Conversión de string a número en coma flotante

El número siempre se interpreta como decimal, independientemente de cualquier prefijo que le pongamos (0 para octal ó 0x para hexadecimal).

La diferencia con el anterior, además de admitir decimales, es que los números en coma flotante admiten la notación exponencial, del tipo "314e-2" o "0.0314e+2".

Si la función encuentra un carácter que no sea un número (0-9), un signo (+ o -), un punto decimal o un exponente, ignorará todos los caracteres que vengan a continuación.

Algunos ejemplos;

```
parseFloat("3.14"); // 3.14
parseFloat("314e-2"); // 3.14
parseFloat("0.0314E+2"); // 3.14
parseFloat("3.14dieciseis"); // 3.14
parseFloat("A3.14"); // NaN
parseFloat("tres"); // NaN
parseFloat("e-2"); // NaN
parseFloat("0x10"); // 0 -> No admite el prefijo 0x para indicar 'hexadecimal'
parseFloat(""); // NaN -> ¡¡el string vacío se convierte a NaN!!
parseFloat(null); // NaN
```

Esta función es muy útil para convertir valores en pixels o puntos, de CSS, en valores numéricos:

```
parseFloat("5px"); // 5
```

Number()

A diferencia de los dos métodos anteriores, éste y el siguiente (conversión implícita) son específicamente para conversión de tipos. En cambio, `parseInt()` y `parseFloat()` son para extraer un número de un string.

`Number()` es un constructor para crear objetos de tipo `Number`, pero cuando se utiliza sin el `new` funciona como un conversor a tipo numérico.

Ejemplos:

```
// como constructor:  
var myNumber = new Number(14);  
  
// como método:  
var myNumber = Number("14");
```

El segundo uso, sin `new`, es el que nos interesa para este tema.

Puede utilizarse para números enteros o decimales y acepta también la notación exponencial.

```
Number("12"); // 12
Number("3.14"); // 3.14
Number("314e-2"); // 3.14
Number("0.0314E+2"); // 3.14
Number("e-2"); // NaN
Number('0x10'); // 16 admite el prefijo 0x para indicar 'hexadecimal'
```

Ignora los espacios al principio y al final, pero, a diferencia de los métodos anteriores, cuando un string contiene caracteres no convertibles a números el resultado siempre es NaN, no trata de 'extraer' la parte numérica.

```
Number(" 12 "); // 12
Number("20 dias"); // NaN
Number("Hace 20 dias"); // NaN
Number("44aa33bb"); // NaN
Number(""); // 0 -> ¡¡el string vacío se convierte a 0!!
Number(" "); // 0
Number(null); // 0
```

Con Number() podemos convertir booleanos en números, false siempre se convierte en 0 y true en 1.

```
Number(true); // 1
Number(false); // 0
```

También podemos incluir una expresión con resultado booleano.

```
Number( (1<2) ); // 1  
Number( (1===2) ); // 0
```

Conversión implícita '+'

La conversión implícita es una forma de conversión rápida a número. Podemos utilizar cualquier operación que fuerce al intérprete a realizar una conversión implícita de tipos pero que no varíe el operando:

```
var myNumberValue = "8" - 0; // number 8  
var myNumberValue = "8" * 1; // number 8  
var myNumberValue = "8" / 1; // number 8  
var myNumberValue = +"8"; // number 8
```

La forma más utilizada por su simplicidad es +var. El operador unitario + no cambia el valor de var pero lo convierte a número. No confundir con ++var que sí cambia el valor, sumándole uno.

Este tipo de conversión, igual que Number(), devuelve NaN si el string contiene caracteres no numéricos.

Como veremos en los siguientes ejemplos, esta forma de conversión es equivalente a Number() y devuelve los mismos resultados:

```
+ "12"; // 12
+ "3.14"; // 3.14
+ "314e-2"; // 3.14
+ "0.0314E+2"; // 3.14
+ "e-2"; // NaN
+ "0x10"; // 16 admite el prefijo 0x para indicar 'hexadecimal'
+ " 12 "; // 12
+ "20 dias"; // NaN
+ "Hace 20 dias"; // NaN
+ "44aa33bb"; // NaN
+ ""; // 0 -> ¡¡el string vacío se convierte a 0!!
+ " "; // 0
+ null; // 0

//boolean
+ true; // 1
+ false; // 0

//también podemos incluir una expresión con resultado boolean
+ (1 < 2); // 1
+ (1 === 2); // 0
```

En resumen:

`parseInt()` tiene un parámetro extra para indicar la base del número (radix).

`parseFloat()` no admite radix. Todos los números se consideran en base decimal.

`parseInt()`, `Number()` y `'+'` interpretan el prefijo `'0x'` como número hexadecimal, `parseFloat()` no.

`parseInt()` y `parseFloat()` pueden extraer un número al principio de un string.

Si el string contiene caracteres no numéricos, `Number()` y `parseFloat()` no lo convierten, devuelve NaN.

Cuando el argumento es un objeto, `parseInt()` y `parseFloat()` llamarán al método `.toString()` antes de analizar la cadena. `Number()` y `parseFloat()` llamarán primero a `.valueOf()` y después a `.toString()` si es necesario.

`parseInt()` no entiende la notación exponencial, todos los demás si.

`parseInt()` y `parseFloat()` convierten el string vacío en NaN.

`Number()` y `parseFloat()` convierten el string vacío en 0.

`parseInt()` y `parseFloat()` de un boolean es NaN.

`Number()` y `parseFloat()` de un boolean devuelve 0 para false y 1 para true.