



## Capítulo 2

# JavaScript

### 2.1. ¿Qué es JavaScript?

<https://es.wikipedia.org/wiki/JavaScript>

JavaScript es un lenguaje de programación que se utiliza principalmente para crear páginas web dinámicas, es decir, aplicaciones web.

Algunas de las principales características de JavaScript son:

- JavaScript es un **lenguaje interpretado**, es decir, no requiere compilación. El navegador del usuario se encarga de interpretar las sentencias JavaScript contenidas en una página HTML y ejecutarlas adecuadamente.
- JavaScript es un **lenguaje orientado a eventos**. Cuando un usuario pincha sobre un enlace o mueve el puntero sobre una imagen se produce un evento. Mediante JavaScript se pueden desarrollar scripts que ejecuten acciones en respuesta a estos eventos.

---

```
1
2 // Cuando hagan click sobre un elemento llamado boton ejecuta la
   función fsuma
3
4 document.getElementById('boton').addEventListener('click', fsuma);
```

---

- JavaScript es un **lenguaje orientado a objetos y basado en prototipos**.

## 2.2. Historia

<https://www.youtube.com/watch?v=iXf900stEK4> (8:57)

Tal y como hemos visto en el vídeo la creación de JavaScript se le atribuye a Brendan Eich, matemático y tecnólogo estadounidense. Eich fue contratado por Netscape Communications Corporation, la empresa que desarrollaba el navegador más difundido de la época: Netscape. Dentro de Netscape y como forma de dar respuesta a la mejora de la experiencia de navegación y velocidad de operación de las páginas web fue creado JavaScript en 1995.

La adopción del nombre JavaScript se atribuye a motivos de marketing, tratando de aprovechar que Java era un lenguaje de programación que estaba adquiriendo gran popularidad y que un nombre similar podía hacer que el nuevo lenguaje fuera atractivo. Pero salvando algunas similitudes, ambos lenguajes son bien distintos. Su principal parecido podemos decir que es el nombre y algunos aspectos de sintaxis, ya que su finalidad y filosofía son muy distintos.

A finales de 1996 Netscape informó que traspasaba la responsabilidad de definir y estandarizar JavaScript a una organización internacional para el desarrollo de estándares en el sector de tecnologías de la información y comunicación denominada Ecma International. Esta organización tuvo su origen en la European Computer Manufacturers Association (ECMA, o Asociación Europea de fabricantes de computadores). En realidad ya no se trata de una organización europea, sino plenamente internacional, que se define como un organismo que asocia a productores de tecnologías de información y comunicación.

Ecma International es el organismo actualmente responsable de la definición de la especificación oficial JavaScript, a la que se denomina ECMAScript. En Ecma International participan grandes empresas relacionadas con internet, computadores y telecomunicaciones, entre las que podemos citar Microsoft, Apple, Google, Yahoo, Toshiba, IBM, Hitachi, Fujitsu, Intel, AMD, Adobe Systems, eBay, Hewlett Packard, Konica Minolta, Sony, etc. y también universidades e instituciones sin ánimo de lucro como Stanford University, UEC Tokyo University, Mozilla Foundation y otras.

ECMAScript fue desarrollado por Ecma International después de que la organización adoptó JavaScript. La primera edición de ECMAScript fue lanzada en 1997.

La especificación oficial de JavaScript (ECMA-262) puede leerse o descargarse en formato pdf accediendo a la dirección web <http://www.ecma-international.org/>

Año	Nombre	Descripción
1997	ECMAScript 1	Primera edición
1998	ECMAScript 2	Sólo cambios editoriales
1999	ECMAScript 3	Añade expresiones regulares y try catch
	ECMAScript4	Nunca fue lanzado
2009	ECMAScript 5	Añadido modo estricto y soporte JSON
2011	ECMAScript 5.1	Cambios editoriales
2015	ECMAScript 6 o 2015	Añadido clases y módulos <a href="https://devcode.la/tutoriales/novedades-ecmascript-6">https://devcode.la/tutoriales/novedades-ecmascript-6</a>
2016	ECMAScript 7 o 2016	Se agregó el operador exponencial (**) y Array.prototype.includes <a href="http://blog.enriqueoriol.com/2016/11/es2016.html">http://blog.enriqueoriol.com/2016/11/es2016.html</a>
2017	ECMAScript 8 o 2017	Funciones asíncronas. Memoria compartida y átomos. Pequeñas novedades en Object.
2018	ECMAScript 9 o 2018	Iteración asíncrona <a href="https://pablomagaz.com/blog/las-novedades-de-ecmascript-9">https://pablomagaz.com/blog/las-novedades-de-ecmascript-9</a>

## 2.3. ¿ Cómo incluir JavaScript en documentos HTML?

La integración de JavaScript y HTML es muy flexible, ya que existen al menos tres formas para incluir código JavaScript en las páginas web.

### 2.3.1. Incluir JavaScript en el mismo documento HTML

El código JavaScript se encierra entre etiquetas **script** y se incluye en cualquier parte del documento aunque se recomienda definir el código JavaScript dentro de la cabecera del documento (dentro de la etiqueta head). Si así lo hacemos, primero se ejecutará el código js y luego se cargará la página.

Si incluimos las etiquetas script en el body; primero se cargará la página y luego se ejecutará el código.

Se pueden poner tantos bloques de código como necesitemos.

Mediante el atributo **language** se debe indicar si se trata de un script de JavaScript o de cualquier otro lenguaje (si no se indica se sobreentiende que es un script de JavaScript).

```
1
2
3 <html xmlns="http://www.w3.org/1999/xhtml">
4 <head>
5 <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
6 <title>Ejemplo de código JavaScript en el propio documento</title>
7 <script>
8     alert("Un mensaje de prueba uno ");
9 </script>
10 </head>
11 <body>
12     <p>Un párrafo de texto.</p>
13     <script>
14         alert("Un mensaje de prueba dos ");
15     </script>
16 </body>
17 </html>
```

Podemos encontrar más información sobre la etiqueta script en páginas del tipo <http://www.virtualnauta.com/es/html/html-etiqueta.php?e=script> y [http://www.w3schools.com/tags/tag\\_script.asp](http://www.w3schools.com/tags/tag_script.asp)

### 2.3.2. Definir JavaScript en un archivo externo

Las instrucciones JavaScript se pueden incluir en un archivo externo de tipo JavaScript que los documentos HTML enlazan mediante la etiqueta script.

Se pueden crear todos los archivos JavaScript que sean necesarios y cada documento HTML puede enlazar tantos archivos JavaScript como necesite.

Este método requiere definir el atributo **src**, que es el que indica la dirección del archivo JavaScript que se quiere enlazar. Cada etiqueta script solamente puede enlazar un único archivo, pero en una misma página se pueden incluir tantas etiquetas script como sean necesarias.

Los archivos de tipo JavaScript son documentos normales de texto con la extensión .js que se pueden crear con cualquier editor de texto.

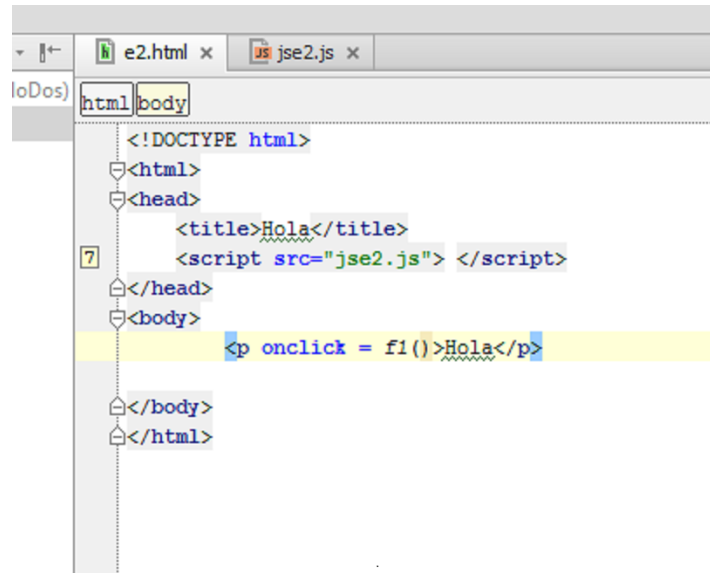
La principal ventaja de enlazar un archivo JavaScript externo es que se simplifica el código HTML de la página, que se puede reutilizar el mismo código JavaScript en todas las páginas del sitio web y que cualquier modificación realizada en el archivo JavaScript se ve reflejada inmediatamente en todas las páginas que lo enlazan.

**Ejemplo:**Archivo jse2.js

---

```
1 function f1(){
2     alert("hola");
3 }
4
```

---

**2.3.3. Incluir JavaScript en los elementos HTML**

Este último método es el menos utilizado, ya que consiste en incluir trozos de JavaScript dentro del código de la página:

---

```
1 <html>
2 <head>
3 <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
4 <title>Ejemplo de código JavaScript en el propio documento</title>
5 </head>
6
7 <body>
8   <p onclick="alert('Un mensaje de prueba')">Un párrafo de texto.</p>
9 </body>
10
11 </html>
```

---

El mayor inconveniente de este método es que ensucia innecesariamente el código de la página y complica el mantenimiento del código JavaScript.

#### 2.3.4. Etiqueta noscript

Algunos navegadores no disponen de soporte completo de JavaScript, otros navegadores permiten bloquearlo parcialmente e incluso algunos usuarios bloquean completamente el uso de JavaScript porque creen que así navegan de forma más segura.

En estos casos, es habitual que si la página web requiere JavaScript para su correcto funcionamiento, se incluya un mensaje de aviso al usuario indicándole que debería activar JavaScript para disfrutar completamente de la página.

---

```
1 <noscript>
2     <p>La página que estás viendo requiere JavaScript</p>
3 </noscript>
```

---

```
1 <head> ... </head>
2 <body>
3
4 <noscript>
5 <p>Bienvenido a Mi Sitio</p>
6 <p>La página que estás viendo requiere para su funcionamiento el uso de
   JavaScript.</p>
7 </noscript>
8
9 </body>
```

---

```
1 <script type="text/javascript">
2     document.write("Hello World!");
3 </script>
4 <noscript>Tu navegador no soporta javascript</noscript>
```

---

La etiqueta noscript se debe incluir en el interior de la etiqueta body (normalmente se incluye al principio). El mensaje que muestra noscript puede incluir cualquier elemento o etiqueta HTML.

[http://www.w3schools.com/tags/tag\\_noscript.asp](http://www.w3schools.com/tags/tag_noscript.asp)

## 2.4. El lenguaje JavaScript

La sintaxis de JavaScript es muy similar a la de otros lenguajes de programación como Java y C. Las normas básicas que definen la sintaxis de JavaScript son las siguientes:

- No se tienen en cuenta los espacios en blanco y las nuevas líneas. El intérprete de JavaScript ignora cualquier espacio en blanco sobrante, por lo que el código se puede ordenar de forma adecuada para entenderlo mejor (tabulando las líneas, añadiendo espacios, creando nuevas líneas, etc.)
- Se distinguen las mayúsculas y minúsculas.
- **No se define el tipo de las variables.** Al crear una variable no es necesario indicar el tipo de dato que almacenará. Una misma variable puede almacenar diferentes tipos de datos durante la ejecución del script.
- No es necesario terminar cada sentencia con el carácter de punto y coma (;). En la mayoría de lenguajes de programación es obligatorio terminar cada sentencia con este carácter. Aunque no es obligatorio, es conveniente seguir la tradición de terminar cada sentencia con el carácter del punto y coma.
- Se pueden y deben incluir comentarios. Aunque el contenido de los comentarios no se visualiza por pantalla, sí que se envía al navegador del usuario junto con el resto del script. JavaScript define dos tipos de comentarios: los de una sola línea y los que ocupan varias líneas.

Los comentarios de una sola línea se definen añadiendo dos barras oblicuas (//) al principio de la línea.

---

```
1
2 // Se muestra un mensaje a través de la función alert
3 alert("mensaje de prueba");
```

---

Los comentarios multilínea se definen encerrando el texto del comentario entre los símbolos /\* y \*/.

---

```
1
2 /* Los comentarios de varias líneas son muy útiles cuando se
   necesita incluir mucha información en los comentarios */
3
4 alert("mensaje de prueba");
```

---



## 2.5. Variables

Una variable es un elemento que se emplea para almacenar y hacer referencia a un valor. Las variables en JavaScript se crean mediante la palabra reservada **var**.

---

```
1 var numero_1 = 3;  
2 var numero_2 = 1;  
3 var resultado = numero_1 + numero_2;
```

---

Una de las características más sorprendentes de JavaScript para los programadores habituados a otros lenguajes de programación es que **no es necesario declarar las variables**. En otras palabras, se pueden utilizar variables que no se han definido anteriormente mediante la palabra reservada **var**. El ejemplo anterior también es correcto en JavaScript de la siguiente forma:

---

```
1 var numero_1 = 3;  
2 var numero_2 = 1;  
3 resultado = numero_1 + numero_2;
```

---

La variable resultado no está declarada, por lo que JavaScript crea una variable global y le asigna el valor correspondiente. En cualquier caso, se recomienda declarar todas las variables que se vayan a utilizar.

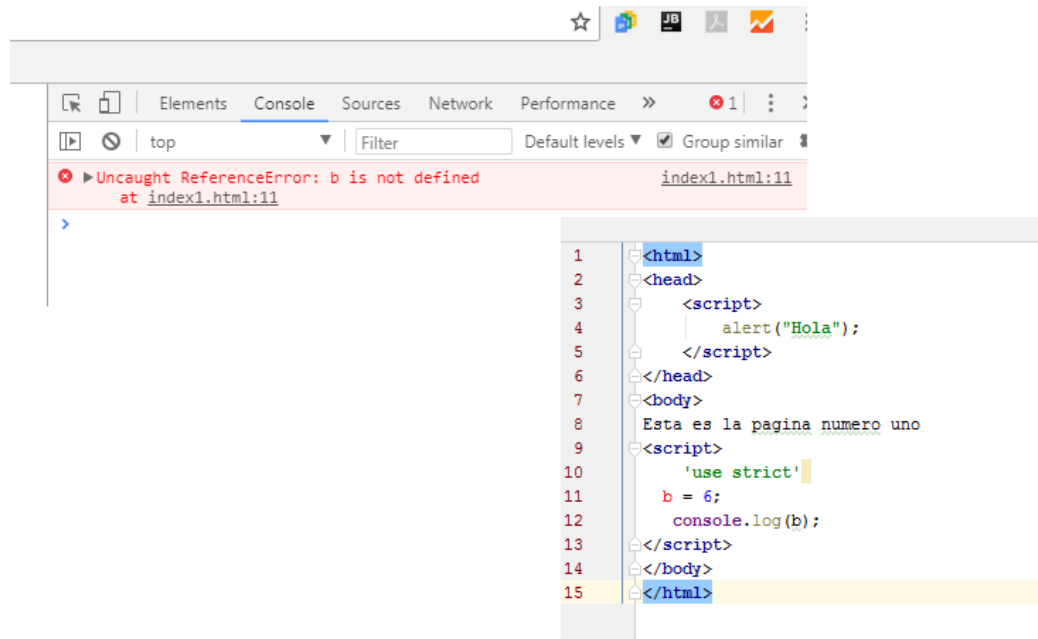
Las variables declaradas se limitan al contexto de ejecución en el cual son declaradas. Las variables no declaradas siempre son globales.

'**use strict**' es una funcionalidad que apareció en ECMAScript 5 que te permite poner un programa (o función) en un contexto estricto.

El modo estricto nos ayuda con factores como:

- Captura algunos errores comunes en nuestro código y nos muestra la excepción.
- Previene (o muestra errores) cuando tomamos algunas acciones consideradas como inseguras en nuestro código.
- Deshabilita funcionalidades que son confusas como el permitir no declarar las variables.

O sea, usar **use strict** es hacer nuestro código mejor y más seguro.



El **nombre de una variable** también se conoce como identificador y debe cumplir las siguientes normas:

- Sólo puede estar formado por letras, números y los símbolos dolar y guión bajo.
- El primer carácter no puede ser un número.

---

```
1 // Correctos
2 var $numero1;
3 var _$letra;
4 var $$otroNumero;
5 var $_a__$4;
6
7 // Incorrectos
8 var 1numero; // Empieza por un número
9 var numero;1_123; // Contiene un carácter ";"
```

---

### 2.5.1. Tipos de variables

Aunque todas las variables se crean de la misma forma (mediante la palabra reservada `var`), la manera en la que se les asigna un valor determina el tipo de dato que almacena (números, textos, etc.)

#### Numéricas

Se utilizan para almacenar valores numéricos enteros o decimales. El valor se asigna indicando directamente el número entero o decimal. Los números decimales utilizan el carácter `.` (punto) en vez de `,` (coma) para separar la parte entera y la parte decimal:

---

```
1 var iva = 16; // variable tipo entero
2 var total = 234.65; // variable tipo decimal
3 var numero=12e5; // Notación científica.
4
5 var cantidad = new Number();
```

---

También podemos crear **objetos de tipo `Number`** en vez de variables numéricas. El valor del objeto `Number` que se crea depende de lo que reciba el constructor de la clase `Number`. Si el constructor recibe un número, entonces inicializa el objeto con el número que recibe. Si recibe un número entrecomillado lo convierte a valor numérico, devolviendo también dicho número. Devuelve 0 en caso de que no reciba nada. En caso de que reciba un valor no numérico devuelve `NaN`, que significa "Not a Number" (No es un número). Si recibe `false` se inicializa a 0 y si recibe `true` se inicializa a 1.

[https://www.w3schools.com/jsref/jsref\\_obj\\_number.asp](https://www.w3schools.com/jsref/jsref_obj_number.asp)

### Cadenas de texto

Se utilizan para almacenar caracteres, palabras y/o frases. Para asignar el valor a la variable, se encierra entre comillas dobles o simples.

---

```
1 var mensaje = "Bienvenido a nuestro sitio web";
2 var nombreProducto = 'Producto ABC';
3 var letraSeleccionada = 'c';
```

---

En ocasiones, el texto que se almacena en las variables no es tan sencillo. Si el propio texto contiene comillas simples o dobles, la estrategia que se sigue es la de encerrar el texto con las comillas (simples o dobles) que no utilice el texto.

---

```
1 /* El contenido de texto1 tiene comillas simples, por lo que se
   encierra con comillas dobles */
2
3
4 var texto1 = "Una frase con 'comillas simples' dentro";
5
6
7 /* El contenido de texto2 tiene comillas dobles, por lo que se
   encierra con comillas simples */
8
9
10 var texto2 = 'Una frase con "comillas dobles" dentro';
```

---

No obstante, a veces las cadenas de texto contienen tanto comillas simples como dobles. Además, existen otros caracteres que son difíciles de incluir en una variable de texto (tabulador, ENTER, etc.) Para resolver estos problemas, JavaScript define un mecanismo para incluir de forma sencilla caracteres especiales y problemáticos dentro de una cadena de texto. El mecanismo consiste en sustituir el carácter problemático por una combinación simple de caracteres. En la imagen se muestra la tabla de conversión que se debe utilizar:

Utilizando estos caracteres, el ejemplo anterior que contenía comillas simples y dobles dentro del texto se puede rehacer de la siguiente forma:

---

```
1 var texto1 = 'Una frase con \'comillas simples\' dentro';
2 var texto2 = "Una frase con \"comillas dobles\" dentro";
```

---

Este mecanismo de JavaScript se denomina **mecanismo de escape** de los caracteres problemáticos y es habitual referirse a que los caracteres han sido escapados.

Otra manera de crear cadenas de texto consiste en instanciar directamente la clase String. La diferencia entre tener una variable y tener

Si se quiere incluir...	Se debe incluir...
Una nueva línea	\n
Un tabulador	\t
Una comilla simple	\'
Una comilla doble	\"
Una barra inclinada	\\

un objeto, como ya sabemos, está en que en una variable sólo tiene un valor; un objeto puede tener varias propiedades además de métodos.

[https://www.w3schools.com/jsref/jsref\\_obj\\_string.asp](https://www.w3schools.com/jsref/jsref_obj_string.asp)

De acuerdo con el estándar ECMA, las cadenas se convierten automáticamente en objetos String cuando intentas llamar a un método por lo que esta manera de crear cadenas no suele ser la más utilizada.

---

```
1 var nombre = new String();
2 nombre = "pepe";
3
4 var nombre4 = new String("Ana");
```

---

### 2.5.2. Los operadores typeof e instanceof

Si tenemos en cuenta que una misma variable, en distintos momentos puede almacenar diferentes tipos de datos, cabe la posibilidad de que en un momento determinado necesitemos conocer no el dato, sino el tipo de datos que almacena.

El operador typeof devuelve una cadena que indica el tipo del dato almacenado en una variable.

---

```
1
2 // En javascript una función es una variable
3 var miFuncion = new Function("5+2");
4 // string
5 var forma="redonda";
6 // número
7 var tamano=1;
8 // Fecha
9 var hoy=new Date();
10 // variable a la que no se ha asignado ningún valor
11 var nombre;
```

---

```
12
13
14 typeof miFuncion == 'function'
15 typeof forma == 'string'
16 typeof tamano == 'number'
17 typeof hoy == 'object'
18 typeof nombre == 'undefined'
```

---

El operador `instanceof` devuelve `true` si un objeto es una instancia de otro especificado.

---

```
1 // array con tres elementos de tipo string
2 var animales = ["perro", "gato", "hamster"];
3
4 alert (animales instanceof Object); //Devuelve true
5 alert (animales instanceof Array); //Devuelve true
6 alert (animales instanceof Number); //Devuelve false*/
7
8 color1=new String("verde")
9 color1 instanceof String // devuelve verdadero (true)
10
11 color2="coral"
12 color2 instanceof String // devuelve falso (color2 no es un objeto
    String)
```

---

### 2.5.3. Conversión

Los números pueden convertirse fácilmente en strings, pero los strings no se pueden convertir directamente en números, para eso existen funciones explícitas de conversión.

---

```
1 var n1 = "7";
2 var n2 = 3;
3 var resultado = n1 + n2;
4 console.log(resultado); // Muestra 73 trata n2 como una cadena sin
    llevar a cabo ninguna conversión explícita.
```

---

**parseFloat(cadena)** transforma en número una cadena de caracteres. Si el número se puede interpretar como número decimal, la función lo tiene en cuenta. Devuelve NaN (Not a Number) cuando la cadena de caracteres empieza con caracteres que no se pueden interpretar como parte de un número. Si la cadena contiene caracteres no numéricos hacia el final, el número se interpreta sólo hasta donde empiezan los caracteres no numéricos, y se devuelve como valor la parte interpretada como número.

**parseInt(cadena)** funciona de forma similar a la función anterior, pero el valor devuelto siempre es un entero.

---

```
1 var dato = "3.14";
```

```
2
3     console.log(parseInt(dato)); // 3
4     console.log(parseFloat(dato)); // 3.14
5
6     dato = "Hola 3.15";
7     console.log(parseInt(dato)); // NaN
8     console.log(parseFloat(dato)); // NaN
9
10    dato = "3.14 Hola";
11    console.log(parseInt(dato)); // 3
12    console.log(parseFloat(dato)); // 3.14
```

---

También existe **toString** para convertir números en cadenas de caracteres.

---

```
1 a = a+' ' // This converts a to string
2 b += ' ' // This converts b to string
3
4 5.41 + ' ' // Result: the string '5.41'
5 Math.PI + ' ' // Result: the string '3.141592653589793'
6
7
8 a = a.toString() // This converts a to string
9 b = b.toString() // This converts b to string
10
11 (5.41).toString() // Result: the string '5.41'
12 (Math.PI).toString() // Result: the string '3.141592653589793'
```

---

A través de **Number** también podemos llevar a cabo conversiones aunque **parseInt** realiza más esfuerzos tal y como se ve en la siguiente imagen. Este objeto también se puede usar para muchas más cosas.

```

<head>
</head>
<body>
  Esta es la pagina numero uno
<script>
  var dato = "3";
  console.log(dato + 2); //32
  console.log(parseInt(dato) + 2); // 5
  console.log(Number(dato) + 2); // 5

  dato = "Hola";
  console.log(parseInt(dato) + 2); // NaN Not a Number
  console.log(Number(dato) + 2); // NaN

  dato = "2a";
  console.log(parseInt(dato) + 2); // 4
  console.log(Number(dato) + 2); // NaN

Number.i
  isFinite(number) (Number) boolean
  isInteger(number) (Number) boolean
  isNaN(number) (Number) boolean
  isSafeInteger(number) (Number) boolean
  isPrototypeOf([Object] o) (Object) boolean
  icon (CSSStyleDeclaration)
  id (several definitions)

```



#### 2.5.4. Booleanos

Una variable de tipo boolean almacena un tipo especial de valor que solamente puede tomar dos valores: true (verdadero) o false (falso).

```
1 var clienteRegistrado = false;  
2 var ivaIncluido = true;  
3  
4 var x = new Boolean();
```

Más información en [https://www.w3schools.com/jsref/jsref\\_obj\\_boolean.asp](https://www.w3schools.com/jsref/jsref_obj_boolean.asp)

#### 2.5.5. Asignación

El operador de asignación es el más utilizado y el más sencillo. Este operador se utiliza para guardar un valor específico en una variable. El símbolo utilizado es = (no confundir con el operador == que se verá más adelante).

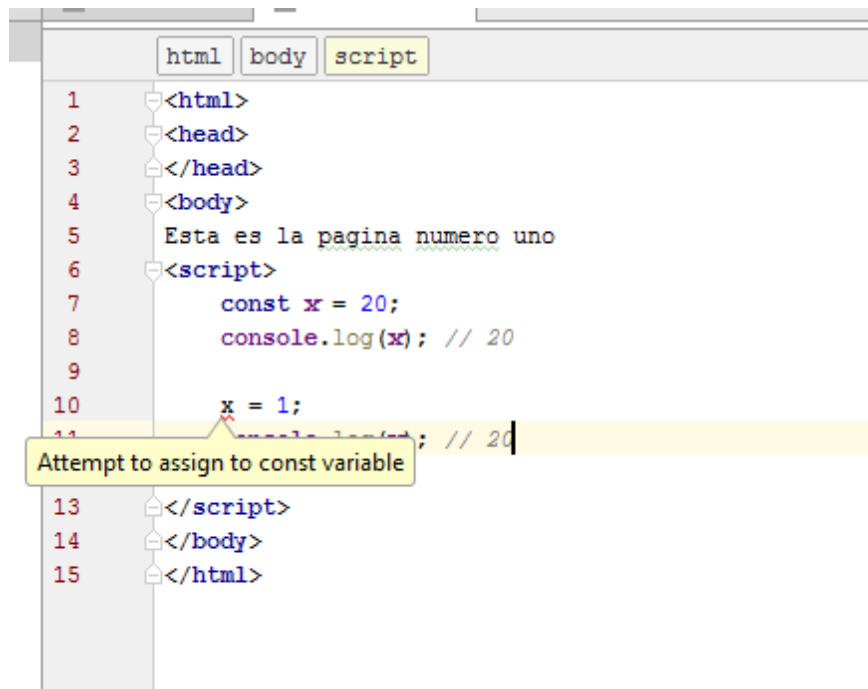
```
1 var numero1 = 3;
```

A la izquierda del operador, siempre debe indicarse el nombre de una variable. A la derecha del operador, se pueden indicar variables, valores, condiciones lógicas, etc...

#### 2.5.6. Definición de variables en ECMAScript 6

Las nuevas formas de definir variables que acompañan a var, son **let** (para controlar mejor el ámbito de la variable) y **const** que trae las constantes en JavaScript.

A las constantes en JavaScript siempre se las ha echado en falta y nos hemos visto obligados a crear pseudo-constantes simplemente estableciendo el nombre de la variable en mayúscula (var PSEUDOCONSTANTE) y dando por hecho que nadie la modificará posteriormente, algo que se podía hacer sin problemas en JavaScript, y ahora como debe de ocurrir en una constante, ya no variará el propio valor por mucho que se intente modificar.



```

1  <html>
2  <head>
3  </head>
4  <body>
5    Esta es la pagina numero uno
6  <script>
7      const x = 20;
8      console.log(x); // 20
9
10     x = 1;
11     console.log(x); // 20
12
13 </script>
14 </body>
15 </html>

```

Attempt to assign to const variable

Como ya hemos dicho JavaScript es un lenguaje bastante flexible, sobre todo con el uso y declaración de variables, que pueden contener cualquier tipo de dato y mutar el tipo sin ningún problema. Además de poder ser declaradas de diferentes formas y pasar a ser globales o locales según como se haga (lo que puede llevar a errores básicos de programación por no declarar correctamente una variable).

Esta nueva versión de JavaScript incorpora `let` para declarar variables y controlar mejor su propagación. En principio es bastante simple de entender, simplemente evita que una variable sea visible cuando no debe como ocurre con `var`.

```

1
2     var a = 0;
3     do
4     {
5         .....
6         var b = 1;
7         .....
8     }
9     while ( a != 0);
10
11     // Fuera del do while no debería existir la variable b
12     console.log(b); // 1
13

```

```
14
15     let a = 0;
16     do
17     {
18         .....
19         let b = 1;
20         .....
21     }
22     while ( a != 0);
23
24     console.log(b); // b is not defined
```

---

## 2.6. Operaciones de entrada/salida

### 2.6.1. alert()

La función alert() permite mostrar al usuario información literal o el contenido de variables en una ventana independiente. La ventana contendrá la información a mostrar y el botón Aceptar.

Su sintaxis es: alert(mensaje)

---

```
1
2 <SCRIPT LANGUAGE="Javascript">
3   alert("Bienvenido");
4 </SCRIPT>
```

---

```
1
2 <SCRIPT LANGUAGE="Javascript">
3   var nombre="María";
4   alert(nombre + "\n bienvenida");
5 </SCRIPT>
```

---

También existe el método write del objeto document que nos permite escribir en la página.

---

```
1
2 <SCRIPT LANGUAGE="Javascript">
3   document.write("Bienvenido");
4 </SCRIPT>
```

---

También para visualizar mensajes se puede utilizar la consola.

---

```
1 var nombre = "Pepe";
2
3 console.log("Hola");
4 console.log("Hola " + nombre);
5 console.log('Hola' + nombre + ' buenos días');
6 console.log('Hola %s buenos días', nombre);
```

---

%s formatea el valor como un string.

%d o %i formatea el valor como un número entero.

%f formatea el valor como un número de punto flotante.

Por último, podemos utilizar **innerHTML** para escribir en la página.

---

```
1 <body>
2
3 <h1 id="cabecera1"> </h1>
4 <div id = "d1"></div>
5
6 <script>
7   document.getElementById("cabecera1").innerHTML = "Texto";
8   document.getElementById("d1").innerHTML = " <p> parrafo nuevo
   </p>";
```

---

```
9    </script>
10
11 </body>
```

---

### 2.6.2. confirm()

A través de la función confirm() se activa un cuadro de diálogo que contiene los botones Aceptar y Cancelar. Cuando el usuario pulsa el botón Aceptar, este método devuelve el valor true; cuando pulsa Cancelar devuelve el valor false.

Su sintaxis es: confirm(mensaje)

```
1 <SCRIPT LANGUAGE="Javascript">
2   var respuesta;
3   respuesta=confirm ("¿Desea cancelar la subscripción?");
4   document.write("Usted ha contestado que " + respuesta)
5 </SCRIPT>
```

---

### 2.6.3. prompt()

La función prompt() abre un cuadro de diálogo en pantalla que pide al usuario que introduzca un dato. Si se pulsa el botón Cancelar, el valor de devolución es false/null. Pulsando en Aceptar se obtiene el valor true y la cadena de caracteres introducida se guarda para su posterior procesamiento.

Su sintaxis es prompt(pregunta, respuesta), siendo pregunta la información que se muestra y respuesta la información por defecto que la orden tomará de entrada.

Ejemplo: Solicita la provincia, dando como opción por defecto Asturias. Se puede introducir otra provincia a la mostrada. Si se pulsa Aceptar se muestra la provincia introducida y si no se muestra una ventana que indica lo sucedido.

```
1 <SCRIPT LANGUAGE="Javascript">
2   var provincia;
3   provincia=prompt("Introduzca la provincia ","Asturias");
4   document.write("Usted ha introducido la siguiente información
5     "+provincia)
6 </SCRIPT>
```

---

Dentro de las páginas web es muy probable que dispongamos de elementos para llevar a cabo la entrada y/o salida de datos (cajas de texto, etiquetas, etc...). Desde javascript podemos muy fácilmente interactuar con estos elementos a través de document.

---

```
1 <body>
2 <input type="text" id="cajaUno">
3 <input type="button" value="Click me" onclick="funcionUno()">
4
5 <script>
6     function funcionUno() {
7         alert("Has escrito: " +
8             document.getElementById("cajaUno").value);
9         document.getElementById("cajaUno").value="Adios";
10    }
11 </script>
```

---

## 2.7. Operadores

Los operadores se dividen en:

- Operadores de asignación
- Operadores aritméticos
- Operadores relacionales
- Operadores lógicos
- Operadores a nivel de bit.
- Operadores cadenas.

Los operadores pueden ser unitarios o binarios; los primeros precisan un único operando y los segundos, dos.

Los **operadores aritméticos** son binarios o unitarios. Los operadores unitarios modifican el valor al que se aplican y son:

Incremento ++

Disminución –

Menos unitario -

Los operadores unitarios matemáticos pueden colocarse antes (prefijos) o después (sufijos) del operando y su valor varía según esta posición, ya que el operador prefijo modifica el operando antes de utilizar su valor, mientras el operador sufijo modifica el operando después de haber utilizado el valor. Un ejemplo puede facilitar la comprensión:

```
1 x=10;  
2 y=x++;  
3 por lo que y=10 y x=11  
4  
5 x=10;  
6 y=++x;  
7 por lo que y=11 y x=11
```

Los operadores binarios matemáticos no cambian el valor de los operandos, sino que memorizan el resultado en un tercer operando. Incluyen las principales operaciones aritméticas:

Suma +

Resta -

Multipliación \*

División /



Ya hemos visto el **operador de asignación** cuyo signo es el igual (=).

Resto (módulo) %

No es posible utilizar el operador de cálculo del módulo cuando los operandos tienen decimales. El operador de división aplicado a variables de tipo entero produce un resultado truncado en la parte decimal. Si se divide el módulo entre cero, se tendrá una excepción en la ejecución: si la operación excede del límite inferior (underflow), el resultado será cero; si excede del límite superior (overflow), se obtendrá una aproximación.

El operador de asignación puede también ser compactado, es decir, puede combinarse con un operador aritmético. En general, cuando las expresiones son del tipo:

variable = variable operador expresión

pueden cambiarse en:

variable operador = expresión

---

```
1 x += y  x = x + y
2 x -= y  x = x - y
3 x *= y  x = x * y
4 x /= y  x = x / y
5 x %= y  x = x % y
```

---

Con **operador relacional** entendemos la relación que tiene un valor respecto a otro. Se basan en el concepto de verdadero o falso y en todos los casos operan con sólo dos estados diversos (0/1, encendido/apagado, verdadero/falso).

---

```
1 > Mayor que
2
3 >= Mayor o igual
4
5 < Menor que
6
7 <= Menor o igual
8
9 == Igual
10
11 != Distinto
12
13 === Igualdad estricta (no sólo compara valor sino que además compara
14     el tipo).
15
15 !== Desigualdad estricta
```

---

Los operadores relacionales, como ya hemos dicho, producen un resultado 1 si la relación es verdadera, y 0 si la relación es falsa.

Los **operadores lógicos** son muy parecidos a los relacionales, en el sentido de que dan también como salida sólo dos valores que, en este caso, son: 0 si la expresión lógica es verdadera, 1 si es falsa.



AND &&

OR ||

NOT !

Los **operadores sobre cadenas** son:

+ Suma

+= Adjunta

## 2.8. Estructuras de control de flujo

La estructura más utilizada en JavaScript y en la mayoría de lenguajes de programación es la estructura if. Se emplea para tomar decisiones en función de una condición. Su definición formal es:

---

```
1  if(condicion)
2  {
3      ...
4  }
```

---

Si la condición se cumple, es decir, si su valor es true; se ejecutan todas las instrucciones que se encuentran dentro de las llaves. Si la condición no se cumple, es decir, si su valor es false; no se ejecuta ninguna instrucción y el programa continua ejecutando el resto de instrucciones.

---

```
1  var mostrarMensaje = true;
2  if(mostrarMensaje)
3  {
4      alert("Hola Mundo");
5  }
```

---

En el ejemplo anterior, el mensaje si que se muestra al usuario ya que la variable mostrarMensaje tiene un valor de true y por tanto, el programa entra dentro del bloque de instrucciones del if. El ejemplo se podría reescribir también como:

---

```
1  var mostrarMensaje = true;
2  if(mostrarMensaje == true)
3  {
4      alert("Hola Mundo");
5  }
```

---

La comparación del ejemplo anterior suele ser el origen de muchos errores de programación, al confundir los operadores == y =. Las comparaciones siempre se realizan con el operador ==, ya que el operador = solamente asigna valores.

La condición que controla el if puede combinar los diferentes operadores lógicos y relacionales mostrados anteriormente (and, or, not) y así encadenar varias condiciones.

---

```
1  var mostrado = false;
2  if(!mostrado)
3  {
4      alert("Es la primera vez que se muestra el mensaje");
5  }
6
7  var mostrado = false;
8  var usuarioPermiteMensajes = true;
9  if(!mostrado && usuarioPermiteMensajes) {
```

```
10 alert("Es la primera vez que se muestra el mensaje");
11 }
```

---

### 2.8.1. Estructura if...else

```
1 if(condicion)
2 {
3 ...
4 }
5 else {
6 ...
7 }
```

---

Si la condición se cumple se ejecutan todas las instrucciones que se encuentran dentro del if(). Si la condición no se cumple se ejecutan todas las instrucciones contenidas en la rama del else.

```
1 var edad = 18;
2 if(edad >= 18)
3 {
4     alert("Eres mayor de edad");
5 }
6 else
7 {
8     alert("Todavía eres menor de edad");
9 }
```

---

El siguiente ejemplo compara variables de tipo cadena de texto:

```
1 var nombre = "";
2 if(nombre == "")
3 {
4     alert("Aun no nos has dicho tu nombre");
5 }
6 else
7 {
8     alert("Hemos guardado tu nombre");
9 }
```

---

La estructura if o if...else se puede encadenar para realizar varias comprobaciones seguidas:

```
1 if(edad < 12)
2 {
3     alert("Todavía eres muy pequeño");
4 }
5 else
6     if(edad < 19)
7     {
8         alert("Eres un adolescente");
9     }
10 else
11     if(edad < 35)
```

---

```
12     {
13         alert("Aún sigues siendo joven");
14     }
15     else
16     {
17         alert("Piensa en cuidarte un poco más");
18     }
```

---

### 2.8.2. Operador ternario

El operador ternario nos permite escribir una sentencia if-else en una única línea. Este operador está formado por tres partes. La primera es la condición, la segunda que hacer si se cumple la condición y la tercera que hacer si no se cumple la condición. Las distintas partes están separados por ? y : respectivamente.

CONDICION ? ResultadoCierto : ResultadoFalse;

---

```
1 <script>
2
3     var edad = 22;
4
5     if (edad > 18)
6         alert("Puede votar");
7     else
8         alert("No puede votar");
9
10    // Operador ternario
11    // (CONDICION) ? RESULTADO_CIERTO : RESULTADO_FALSO
12    var puedeVotar = (edad > 18) ? "Puede votar" : "No puede
        votar";
13    alert (puedeVotar);
14
15 </script>
```

---

### 2.8.3. Estructuras switch

La instrucción switch es una alternativa para reemplazar los if/else if. De todos modos se puede aplicar en ciertas situaciones donde la condición se verifica si es igual a cierto valor. No podemos preguntar por mayor o menor.

---

```
1 <html>
2 <head></head>
3 <body>
4     <script language="javascript">
5         var valor;
6         valor=prompt('Ingrese un valor comprendido entre 1 y 5:', '');
7         //Convertimos a entero
8         valor=parseInt(valor);
9         switch (valor)
10     {
```

```
11     case 1: document.write('uno');
12         break;
13     case 2: document.write('dos');
14         break;
15     case 3: document.write('tres');
16         break;
17     case 4: document.write('cuatro');
18         break;
19     case 5: document.write('cinco');
20         break;
21     default:
22         document.write('debe ingresar un valor comprendido entre
23             1 y 5.');
```

---

```
23     }
24 </script>
25 </body>
26 </html>
```

Debemos tener en cuenta que la variable que analizamos debe ir después de la instrucción switch entre paréntesis. Cada valor que se analiza debe ir después de la palabra clave 'case'.

Es importante disponer la palabra clave 'break' al finalizar cada caso. La instrucciones que hay después de la palabra clave default se ejecutan en caso que la variable no se verifique en algún case. El default es opcional en esta instrucción.

#### 2.8.4. Estructura for

---

```
1 for(inicializacion; condicion; actualizacion)
2 {
3     ...
4 }
```

---

La idea del funcionamiento de un bucle for es la siguiente: mientras la condición indicada se cumpla, repite la ejecución de las instrucciones definidas dentro del for. Además, después de cada repetición, actualiza el valor de las variables que se utilizan en la condición.

La inicialización es la zona en la que se establece los valores iniciales de las variables que controlan la repetición.

La condición es el único elemento que decide si continua o se detiene la repetición.

La actualización es el nuevo valor que se asigna después de cada repetición a las variables que controlan la repetición.

---

```
1 var mensaje = "Hola, estoy dentro de un bucle";
2 for(var i = 0; i < 5; i++)
3 {
```

```
4         alert(mensaje);
5     }
6
7     // Dos variables.
8     for(x=1,y=10; x <= 10 && y!=0; x++,y--)
```

---

### 2.8.5. Estructura for...in

Una estructura de control derivada de for es la estructura for...in. Su definición exacta implica el uso de objetos.

---

```
1     for(indice in array)
2     {
3         ...
4     }
5
6     var dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes",
7                 "Sábado", "Domingo"];
8
9     for(i in dias)
10    {
11        alert(dias[i]);
12    }
13
14    // Objeto animal con variables propiedades
15    var animal = {nombre:"Pica",raza:".....",edad:2}
16    for(propiedad in animal)
17    {
18        animal[propiedad]
```

---

La variable que se indica como índice es la que se puede utilizar dentro del bucle for...in para acceder a los distintos elementos.

### 2.8.6. Estructura While

Con este tipo de estructuras en primer lugar se verifica la condición (repetitiva de 0 a N), si la misma resulta verdadera se ejecutan las operaciones que indicamos entre las llaves que le siguen. Si la condición es falsa continúa con la instrucción siguiente al bloque de llaves. El bloque se repite MIENTRAS la condición sea Verdadera.

Ejemplo: Realizar un programa que imprima en pantalla los números del 1 al 100.

---

```
1     <html>
2     <head></head>
3     <body>
4     <script language="javascript">
5         var x;
```

---

```
6   x=1;
7   while (x<=100)
8   {
9       document.write(x);
10      document.write('<br>');
11      x=x+1;
12  }
13 </script>
14 </body>
15 </html>
```

---

### 2.8.7. Estructura repetitiva (do/while)

La sentencia do/while es otra estructura repetitiva que ejecuta al menos una vez su bloque repetitivo, a diferencia del while que puede no ejecutar el bloque. Esta estructura repetitiva se utiliza cuando conocemos de antemano que por lo menos una vez se ejecutará el bloque repetitivo.

La condición de la estructura está después del bloque a repetir, a diferencia del while que está en la parte superior. Finaliza la ejecución del bloque repetitivo cuando la condición retorna falso igual que el while y el for.

Ejemplo: Escribir un programa que solicite la carga de un número entre 0 y 999, y nos muestre un mensaje de cuántos dígitos tiene el mismo. Finalizar el programa cuando se cargue el valor 0.

---

```
1  <html>
2  <head></head>
3  <body>
4  <script language="javascript">
5      var valor;
6      do
7      {
8          valor=prompt('Ingrese un valor entre 0 y 999:', '');
9          valor=parseInt(valor);
10         document.write('El valor '+valor+' tiene ');
11         if (valor<10)
12         {
13             document.write('Tiene 1 dígitos');
14         }
15         else
16         {
17             if (valor<100)
18             {
19                 document.write('Tiene 2 dígitos');
20             }
21             else
22             {
23                 document.write('Tiene 3 dígitos');
24             }
25         }
26         document.write('<br>');
27     }
```

```
28     while(valor!=0);  
29 </script>  
30 </body>  
31 </html>
```

---



## 2.9. Arrays

Un array es una variable con múltiples valores que pueden ser todos del mismo tipo **o cada una de un tipo diferente**.

---

```
1 var dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes",  
  "Sábado", "Domingo"];  
2  
3 var arrayMuchasDimensiones = [1, ["hola", "que", "tal", ["estas",  
  "estamos", "estoy"], ["bien", "mal"], "acabo"], 2, 5];  
4  
5 var matriz=new Array(3);  
6  
7 for (i = 0; i < 3; i++)  
8 {  
9   matriz[i]=new Array(3);  
10 }  
11  
12 for(i in dias)  
13 {  
14   document.write(dias[i]);  
15 }
```

---

Para definir un array, se utilizan los caracteres [ y ] para delimitar su comienzo y su final y se utiliza el carácter , (coma) para separar sus elementos:

---

```
1 var nombre_array = [valor1, valor2, ..., valorN];
```

---

Se accede a los elementos que forman un array indicando su posición dentro del mismo teniendo en cuenta que los elementos empiezan a contarse en el 0.

---

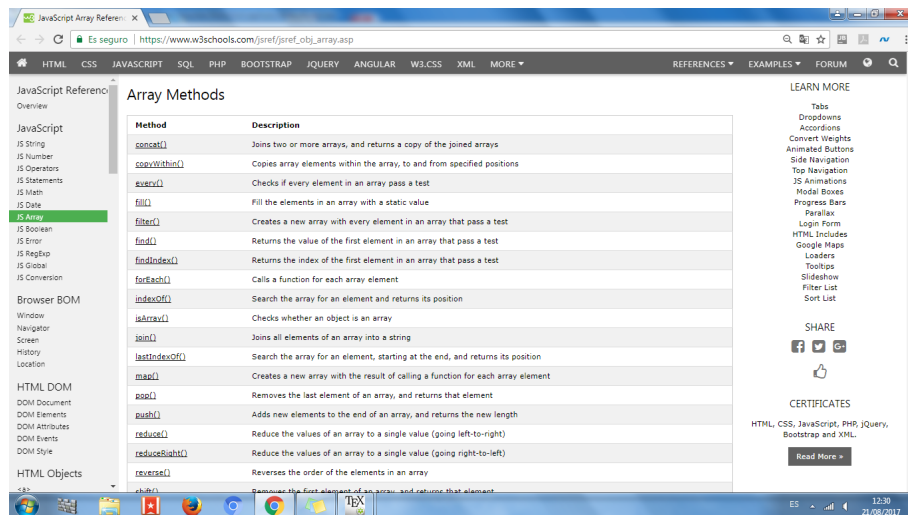
```
1 var diaSeleccionado = dias[0]; // diaSeleccionado = "Lunes"  
2 var otroDia = dias[5]; // otroDia = "Sábado"
```

---

```
1 <html>  
2   <head>  
3     <script>  
4       var array = new Array();  
5       var arrayDos = new Array(2);  
6  
7       array[0] = "Posición ";  
8       array[1] = 0;  
9       alert(array[0] + array[1]);  
10  
11       var temperaturas_cuidades = new Array(5,0,2);  
12  
13     </script>  
14   </head>  
15 </html>
```

---

En la dirección [http://www.w3schools.com/jsref/jsref\\_obj\\_array.asp](http://www.w3schools.com/jsref/jsref_obj_array.asp) podéis encontrar información sobre los distintos métodos codificados en la clase Array.



```

1
2 //ARRAY: almacena en una misma variable múltiples valores: valores
   primitivos, objetos, etc. Se referencian con un índice numérico.
3
4 //OPERACIONES BÁSICAS CON UN ARRAY
5
6 //Crear un array: var nombreArray = [<valores separados por
   comas>];
7   var animales = ["Perro", "Gato", "Hamster"];
8
9 //var nombreArray = new Array(<valores separados por comas>);
10  var animales2 = new Array("Perro", "Gato", "Hamster");
11
12 //Acceso a elementos de un array: nombreArray[<índice>]; Desde el
   0
13  var miAnimal = animales[0];
14
15 //Mostrar todo el array: con una instrucción de mostrar
16  alert(animales);
17  document.write(animales);
18
19 //PROPIEDADES:
20 //length: devuelve la longitud del array (número de elementos)
21  document.write("La longitud del array es: " + animales.length);
22
23 //Mostrar los valores del array uno a uno
24  document.write("<br/><br/>Todos los elementos:")
25  for (var i=0; i<animales.length; i++){
26    document.write("<br/>" + animales[i]);
27  }
28
29 //MÉTODOS:
30 //Array.isArray(<nombreArray>): devuelve true si es un objeto de
   tipo Array. Si pusamos typeof <nombreArray> devolverá object.
31  document.write("<br/>¿Es un array? "+Array.isArray(animales));

```

```

32
33 //<nombreArray> instanceof Array: devuelve true si es un Array.
34 document.write ("<br/>¿Seguro? "+(animales instanceof Array));
35
36 //MÉTODOS PARA MOSTRAR EL ARRAY:
37 //toString(): convierte el array a cadena
38 document.write("<br/>El array en tipo String es: " +
39     animales.toString());
40
41 //join("<separador>"): convierte el array a cadena separado por
42     el separador:
43 document.write("<br/>El array con join es: "+animales.join(" *
44     "));
45
46 //MÉTODOS PARA AÑADIR, EXTRAER O BORRAR ELEMENTOS:
47 //pop() y shift(): extrae el último y primer elemento
48     respectivamente y lo guarda en una variable (si queremos)
49 var ultimo = animales.pop();
50 document.write("<br/>Después de sacar " + ultimo + " quedan
51     "+animales.toString());
52
53 //delete nombreArray[<índice>]: elimina el elemento y lo
54     transforma a undefined
55
56 //push(<elemento>) y unshift(<elemento>): añade un elemento al
57     final y al principio del array respectivamente
58 animales.push("Jilguero");
59 document.write("<br/>Después de meter Jilguero quedan " +
60     animales.toString());
61
62 //Nota: podemos añadirlo con animales[<índice>] = <elemento>;
63     pero hay que tener cuidado de no sobrescribir o dejar huecos.
64
65 //splice(<posicion insertar/borrar>, <elementos a borrar>,
66     [<elementos a añadir separados por comas>]):
67 animales.splice(2,1,"Vaca","Toro");
68 document.write("<br/>Después de meter 2 y borrar 1 quedan " +
69     animales.toString());
70
71 //animales.splice(0,1): eliminaría el primer elemento.
72
73 //slice(<ini>[,<fin>]): devuelve un subarray desde la posición
74     indicada hasta (sin incluir) la final (no es obligatorio)
75 var subarray = animales.slice(1,3);
76 document.write("<br/>El subarray entre 1-3 es: " +
77     subarray.toString());
78
79 //concat(<lista de arrays separados por comas>): une el array
80     inicial con un segundo array
81 var masAnimales = ["Burro", "Mula"];
82 animales.concat(masAnimales);
83 document.write("<br/>El array completo es: " +
84     animales.toString());
85
86 //copyWithin(): copia elementos del array y los sustituye por
87     otros elementos del array.
88 //fill("<elemento>"): sustituye los elementos del array por el
89     indicado.

```

```

74
75 //MÉTODOS PARA BUSCAR:
76 //indexOf(<elemento>[,<pos>]) y lastIndexOf(<elemento>[,<pos>]):
    devuelve la primera o última posición de un elemento
    respectivamente; podemos pasarle a partir de qué elemento
    buscará.
77 document.write("<br/>La primera posición de Toro es: " +
    animales.indexOf("Toro"));
78 document.write("<br/>La última posición de Toro es: "
    +animales.lastIndexOf("Toro"));
79
80 //MÉTODOS PARA ORDENAR E INVERTIR EL ORDEN
81 //reverse(): invierte el orden de un array
82 animales.reverse();
83 document.write("<br/>El array después de invertir es: " +
    animales.toString());
84 //sort(): ordena el array
85 array.sort();
86 document.write("<br/>El array después de ordenar es: " +
    animales.toString());
87 //¡Ojo! Los números almacenados como cadenas se comparan carácter
    a carácter, no como cifras.

```

Una de las últimas novedades de javascript es Array.prototype.includes que determina si un array incluye un elemento determinado.

[https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos\\_globales/Array/includes](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Array/includes)

Destacar el método **push** que nos permite añadir elementos. **Los arrays en javascript son dinámicos.**

También se pueden crear arrays instanciando la clase Array.

```

1 [elemento0, elemento1, ..., elementoN]
2
3 new Array(elemento0, elemento1[, ..., elementoN])
4
5 new Array(longitudDelArray)

```

[https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos\\_globales/Array](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Array)

### 2.9.1. forEach

El método forEach() ejecuta la función indicada una vez por cada elemento del array.

```

1 var array1 = ['a', 'b', 'c'];
2
3 array1.forEach(function(element) {
4     console.log(element);
5 });
6

```

```
7 // "a"  
8 // "b"  
9 // "c"
```

---

## 2.10. Funciones y propiedades básicas de JavaScript

### 2.10.1. Funciones útiles para cadenas de texto

A continuación se muestran algunas de las funciones mas útiles para el manejo de cadenas de texto (clase String). Para más información podemos consultar la dirección [http://www.w3schools.com/jsref/jsref\\_obj\\_string.asp](http://www.w3schools.com/jsref/jsref_obj_string.asp):

- **length**, calcula la longitud de una cadena de texto.
- **+**, se emplea para concatenar varias cadenas de texto. Además del operador +, también se puede utilizar la función **concat()**

---

```
1 var mensaje1 = "Hola";  
2 var mensaje2 = mensaje1.concat(" Mundo"); // mensaje2 = "Hola  
   Mundo"
```

---

- **toUpperCase()** transforma todos los caracteres de la cadena a sus correspondientes caracteres en mayúsculas.
- **toLowerCase()** transforma todos los caracteres de la cadena a sus correspondientes caracteres en minúsculas.
- **charAt(posición)** obtiene el carácter que se encuentra en la posición indicada.
- **indexOf(caracter)** devuelve la posición en la que se encuentra el carácter indicado dentro de la cadena de texto. Si el carácter se incluye varias veces dentro de la cadena de texto, se devuelve su primera posición empezando a buscar desde la izquierda. Si la cadena no contiene el carácter, la función devuelve el valor -1.
- **lastIndexOf(caracter)** devuelve la ultima posición en la que se encuentra el carácter indicado dentro de la cadena de texto. Si la cadena no contiene el carácter, la función devuelve el valor -1.  
Esta función comienza su búsqueda desde el final de la cadena hacia el principio, aunque la posición devuelta es la correcta empezando a contar desde el principio de la palabra.
- **substring(inicio, final)** extrae una porción de una cadena de texto. El segundo parámetro es opcional. Si solo se indica el parámetro inicio, la función devuelve la parte de la cadena original correspondiente desde esa posición hasta el final.  
Si se indica un inicio negativo, se devuelve la misma cadena original.

Cuando se indica el inicio y el final, se devuelve la parte de la cadena original comprendida entre la posición inicial y la inmediatamente anterior a la posición final, es decir, la posición inicio esta incluida y la posición final no.

Si se indica un final más pequeño que el inicio, JavaScript los considera de forma inversa, ya que automáticamente asigna el valor mas pequeño al inicio y el mas grande al final.

- **split(separador)** convierte una cadena de texto en un array de cadenas de texto. La función parte la cadena de texto determinando sus trozos a partir del carácter separador indicado.

---

```
1 var mensaje = "Hola Mundo, soy una cadena de texto!";
2 var palabras = mensaje.split(" ");
3 // palabras = ["Hola", "Mundo,", "soy", "una", "cadena", "de",
4 //             "texto!"];
5
6
7 var palabra = "Hola";
8 var letras = palabra.split(""); // letras = ["H", "o", "l", "a"]
```

---

### 2.10.2. Funciones útiles para números

[https://www.w3schools.com/jsref/jsref\\_obj\\_number.asp](https://www.w3schools.com/jsref/jsref_obj_number.asp)

A continuación se muestran algunas de las funciones y propiedades mas útiles para el manejo de números.

- **NaN**, (del ingles, "Not a Number") JavaScript emplea el valor NaN para indicar un valor numérico no definido (por ejemplo, la división 0/0).
- **isNaN()**, permite proteger a la aplicación de posibles valores numéricos no definidos.

---

```
1 var numero1 = 0;
2 var numero2 = 0;
3 if(isNaN(numero1/numero2))
4 {
5     alert("La division no esta definida para los numeros
6           indicados");
7 }
8 else {
9     alert("La division es igual a => " + numero1/numero2);
10 }
```

---

- **Infinity** hace referencia a un valor numérico infinito y positivo (también existe el valor **Infinity** para los infinitos negativos)

```
1 var numero1 = 10;  
2 var numero2 = 0;  
3 alert(numero1/numero2); // se muestra el valor Infinity
```

- **toFixed(dígitos)**, devuelve el número original con tantos decimales como los indicados por el parámetro dígitos y realiza los redondeos necesarios. Se trata de una función muy útil por ejemplo para mostrar precios.

```
1 var numero1 = 4564.34567;  
2 numero1.toFixed(2); // 4564.35  
3 numero1.toFixed(6); // 4564.345670  
4 numero1.toFixed(); // 4564
```

### 2.10.3. La clase Math

Math posee propiedades y métodos creados por constantes y funciones matemáticas. No tiene constructor.

Propiedades:

- **Math.E** Constante de Euler, la base de los logaritmos naturales, aproximadamente 2.718
- **Math.LN2** Logaritmo natural de 2, aproximadamente 0.693
- **Math.PI**
- .....

Métodos:

- **Math.abs(x)** Devuelve el valor absoluto de un número.
- **Math.acos(x)** Devuelve el arco coseno de un número.
- **Math.cos(x)** Devuelve el coseno de un número.
- **Math.cosh(x)** Devuelve el coseno hiperbólico de un número.
- **Math.exp(x)** Devuelve  $e^x$ , donde  $x$  es el argumento, y  $e$  es la constante de Euler (2.718...), la base de los logaritmos naturales.
- **Math.expm1(x)** Las devoluciones restando 1  $\exp(x)$
- **Math.floor(x)** Devuelve el mayor entero menor que o igual a un número.
- **Math.fround(x)** Devuelve la representación flotante de precisión simple más cercana de un número.



- **Math.hypot([x[, y[, ?]]])** Devuelve la raíz cuadrada de la suma de los cuadrados de sus argumentos.
- **Math.imul(x, y)** Devuelve el resultado de una multiplicación de enteros de 32 bits.
- **Math.log(x)** Devuelve el logaritmo natural (log, también ln) de un número.
- **Math.max([x[, y[, ?]]])** Devuelve el mayor de cero o más números.
- **Math.min([x[, y[, ?]]])** Devuelve el más pequeño de cero o más números.
- **Math.pow(x, y)** Las devoluciones de base a la potencia de exponente, que es, base exponente.
- **Math.random()** Devuelve un número pseudo-aleatorio entre 0 y 1.
- **Math.round(x)** Devuelve el valor de un número redondeado al número entero más cercano.
- **Math.sign(x)** Devuelve el signo de la x, que indica si x es positivo, negativo o cero.
- **Math.sqrt(x)** Devuelve la raíz cuadrada positiva de un número.
- **Math.trunc(x)** Devuelve la parte entera del número x, la eliminación de los dígitos fraccionarios.
- .....

Información más completa la podemos encontrar en [http://www.w3schools.com/js/js\\_math.asp](http://www.w3schools.com/js/js_math.asp) o [https://www.w3schools.com/jsref/jsref\\_obj\\_math.asp](https://www.w3schools.com/jsref/jsref_obj_math.asp)

#### 2.10.4. Clase Date en javascript

[http://www.w3schools.com/jsref/jsref\\_obj\\_date.asp](http://www.w3schools.com/jsref/jsref_obj_date.asp)

Sobre la clase Date recae todo el trabajo con fechas y horas en Javascript. Un objeto de la clase Date se puede crear de dos maneras distintas. Por un lado podemos crear el objeto con la fecha y hora actuales y por otro podemos crearlo con una fecha y hora distintos a los actuales. Esto depende de los parámetros que pasemos al construir el objeto.

Para crear un objeto fecha con el día y hora actuales colocamos los paréntesis vacíos al llamar al constructor de la clase Date.

```
1  
2 miFecha = new Date();
```

Para crear un objeto fecha con un día y hora distintos de los actuales tenemos que indicar entre paréntesis el momento con que inicializar el objeto. Hay varias maneras de expresar una fecha y hora válidas:

---

```
1 miFecha = new Date(año,mes,dia,hora,minutos,segundos);
2
3
4 miFecha = new Date(año,mes,dia);
```

---

Los valores que debe recibir el constructor son siempre numéricos. Un detalle importante, el mes comienza por 0, es decir, enero es el mes 0. Los días de la semana también empiezan en cero (0 Domingo). Si no indicamos la hora, el objeto fecha se crea con hora 00:00:00.

Los objetos de la clase Date no tienen propiedades pero si un montón de métodos.

- **getDate()** Devuelve el día del mes.
- **getDay()** Devuelve el día de la semana.
- **getHours()** Retorna la hora.
- **getMinutes()** Devuelve los minutos.
- **getMonth()** Devuelve el mes (atención al mes que empieza por 0).
- **getSeconds()** Devuelve los segundos.
- **getTime()** Devuelve los milisegundos transcurridos entre el día 1 de enero de 1970 y la fecha correspondiente al objeto al que se le pasa el mensaje.
- **getFullYear()** Retorna el año, al que se le ha restado 1900. Por ejemplo, para el 1995 retorna 95, para el 2005 retorna 105.
- **getFullYear()** Retorna el año con todos los dígitos.
- **setDate()** Actualiza el día del mes.
- **setHours()** Actualiza la hora.
- **setMinutes()** Cambia los minutos.
- **setMonth()** Cambia el mes (atención al mes que empieza por 0).
- **setSeconds()** Cambia los segundos.
- **setTime()** Actualiza la fecha completa. Recibe un número de milisegundos desde el 1 de enero de 1970.
- **setYear()** Cambia el año recibe un número, al que le suma 1900 antes de colocarlo como año de la fecha.

- **setFullYear()** Cambia el año de la fecha al número que recibe por parámetro. El número se indica completo ej: 2005 o 1995. Utilizar este método para estar seguros que todo funciona para fechas posteriores a 2000.

Ejemplo:

```

1  Mostrar en una página la fecha y la hora actual.
2  <HTML>
3  <HEAD>
4
5  <SCRIPT type="text/javascript">
6      function mostrarFechaHora()
7      {
8          var fecha
9          fecha=new Date();
10         document.write('Hoy es ');
11         document.write(fecha.getDate()+' ');
12         document.write((fecha.getMonth()+1)+' / ');
13         document.write(fecha.getFullYear());
14         document.write('<br>');
15         document.write('Es la hora ');
16         document.write(fecha.getHours()+' : ');
17         document.write(fecha.getMinutes()+' : ');
18         document.write(fecha.getSeconds());
19     }
20
21     //Llamada a la función
22     mostrarFechaHora();
23 </SCRIPT>
24
25 </HEAD>
26 <BODY>
27
28 </BODY>
29 </HTML>

```

```

1
2     //Creación de fecha actual:
3     var actual = new Date();
4     alert(actual);
5
6     //Creación de fecha con cadenas:
7     var d1 = new Date("Wed Mar 25 2015 09:23:45 GMT +0100 (W.
8         Europe Standard Time)"); //Ignora si el día de la semana
9         está mal o los paréntesis
10    var d2 = new Date("October 12, 2016 10:30:00");
11    var d3 = new Date("January 25 2015");
12    var d4 = new Date("Jan 25 2015"); //También "25 Jan 2015"
13    //Podemos utilizar comas porque se ignoran
14    var d5 = new Date("2016-05-12T12:34:25");
15    var d6 = new Date("2016-05-12"); // YYYY-MM-DD
16    var d7 = new Date("2016/05/12"); //YYYY/MM/DD o DD/MM/YYYY
17    var d8 = new Date("2016-05"); //El día se sustituye por 1
18    var d9 = new Date("2016"); //El día y el mes se sustituyen por
19        1
20    //¡OJO! Incluimos 0 en días/meses inferiores a 10

```

```
18     alert (d1);
19     alert (d2);
20     alert (d3);
21     alert (d4);
22     alert (d5);
23     alert (d6);
24     alert (d7);
25     alert (d8);
26     alert (d9);
27
28     //Creación de fecha con milisegundos
29     var dms = new Date(864000000);
30     alert ("Fecha en ms: "+dms);
31
32     //Creación de fechas con números
33     var fechaLargo = new Date(2015,11,10,14,30,25);
34     var fechaCorto = new Date(2015,11,10);
35     alert ("Fecha largo: "+fechaLargo+ " Fecha corto:
36         "+fechaCorto);
37
38     var fecha = new Date();
39     alert ("Fecha: "+fecha);
40     alert ("toString: "+fecha.toString());
41     alert ("toUTCString: "+fecha.toUTCString());
42     alert ("toDateString: "+fecha.toDateString());
43
44     //MÉTODOS GET
45     var fc = new Date("October 1, 2016 10:30:20");
46     alert(fc.getDay()+"
47         "+fc.getDate()+"/"+fc.getMonth()+"/"+fc.getFullYear());
48     alert(fc.getHours()+":"+fc.getMinutes()+":"+fc.getSeconds()+":"+fc.getMilliseconds());
49     alert("Ms desde 1/1/1970: "+fc.getTime());
50
51     //MÉTODOS SET
52     fc.setDate(31);
53     fc.setMonth(11);
54     fc.setFullYear(2031);
55     fc.setHours(23);
56     fc.setMinutes(59);
57     fc.setSeconds(59);
58     fc.setMilliseconds(59);
59     //fc.setTime(<ms>); //Modificaríamos la fecha y la hora
60     contando los milisegundos desde el 1/1/1970
61     alert(fc.toString());
62
63     //Ejemplo de cálculo de horas
64     fc.setMonth(fc.getMonth()+3);
65     alert(fc.toString());
```

---

## 2.11. Nuestras funciones

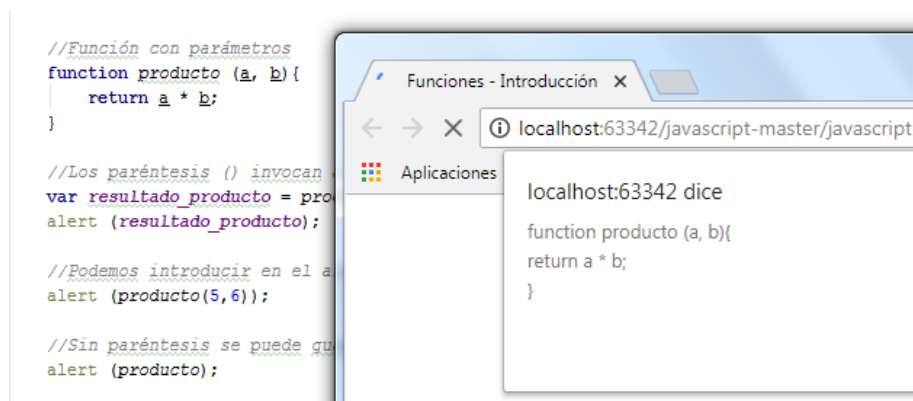
Las estructuras de control, los operadores y todas las utilidades propias de JavaScript que se han visto permiten crear scripts sencillos y de mediana complejidad. Para aplicaciones más complejas son necesarios otros elementos como las funciones.

Como ya sabemos, una función es un conjunto de instrucciones que se agrupan para realizar una tarea concreta y que se pueden reutilizar fácilmente.

Las funciones en JavaScript se definen mediante la palabra reservada **function**, seguida del nombre de la función.

```
1 // Definición
2 function nombre_funcion() {
3   ...
4 }
5
6 // Llamada
7 nombre_funcion();
```

El nombre de la función se utiliza para **llamar a esa función** cuando queramos ejecutarla. Después del nombre de la función, se incluyen dos paréntesis para que se ejecute. Si no se ponen los paréntesis, se está introduciendo la función en una variable.



Por último, las llaves se utilizan para encerrar todas las instrucciones que pertenecen a la función. Recordad que las variables declaradas dentro de estas llaves, serán **locales** a la función, es decir, no se podrán utilizar fuera. Las variables declaradas fuera de las funciones, serán variables **globales** que se podrán utilizar dentro y fuera de las funciones.

Las variables no declaradas son automáticamente globales.

### 2.11.1. Argumentos y valores de retorno

Las variables que necesitan las funciones se llaman parámetros o argumentos. La función debe indicar cuántos argumentos necesita y cual es el nombre de cada uno no el tipo de dato. Al llamar a la función se deben incluir los valores que se le van a pasar. Los argumentos se indican dentro de los paréntesis que van detrás del nombre de la función y se separan con una coma (,).

---

```
1
2 function suma(primerNumero, segundoNumero)
3 {
4     var resultado = primerNumero + segundoNumero;
5     alert("El resultado es " + resultado);
6 }
```

---

Dentro de la función, el valor de la variable primerNumero será igual al primer valor que se le pase a la función en la llamada y el valor de la variable segundoNumero será igual al segundo valor que se le pasa. Para pasar valores a la función, se incluyen dentro de los paréntesis utilizados al llamar a la función.

---

```
1 // Definición de la función
2 function suma(primerNumero, segundoNumero)
3 {
4     var resultado = primerNumero + segundoNumero;
5     alert("El resultado es " + resultado);
6 }
7
8 // Declaración de las variables
9 var numero1 = 3;
10 var numero2 = 5;
11
12 // Llamada a la función
13 suma(numero1, numero2);
```

---

El número de argumentos que se pasa a una función debería ser el mismo que el número de argumentos que se ha indicado pero JavaScript **no muestra ningún error si se pasan más o menos argumentos de los necesarios**.

---

```
1 function sumAll(a,b,c) {
2
3     return a + b + c;
4 }
5
6 var resultado = sumAll(1, 123);
7
8 // El contenido de resultado será NaN
```

---

Si se le pasan más argumentos de los indicados se puede acceder a ellos a través del **array arguments**.

```
1 x = findMax(1, 123, 500, 115, 44, 88);
2
3 function findMax() {
4     var i;
5     var max = 0;
6     for (i = 0; i < arguments.length; i++)
7     {
8         if (arguments[i] > max)
9         {
10             max = arguments[i];
11         }
12     }
13     return max;
14 }
```

---

El orden de los argumentos es fundamental, ya que el primer dato que se indica en la llamada, será el primer valor que espera la función; el segundo valor indicado en la llamada, es el segundo valor que espera la función y así sucesivamente.

Se puede utilizar un número ilimitado de argumentos, aunque si su número es muy grande, se complica en exceso la llamada a la función.

El paso de argumentos se realiza por valor no por referencia.

Las funciones no solamente pueden recibir datos, sino que también pueden devolver el valor que han calculado. Para devolver valores dentro de una función, se utiliza la palabra reservada **return**. Aunque las funciones pueden devolver valores de cualquier tipo, solamente pueden devolver un valor cada vez que se ejecutan.

```
1 function calculaPrecioTotal(precio) {
2     var impuestos = 1.16;
3     var gastosEnvio = 10;
4     var precioTotal = ( precio * impuestos ) + gastosEnvio;
5     return precioTotal;
6 }
7
8 var precioTotal = calculaPrecioTotal(23.34);
9 // Seguir trabajando con la variable "precioTotal"
```

---

Para que la función devuelva un valor, solamente es necesario escribir la palabra reservada **return** junto con el nombre de la variable que se quiere devolver o el valor.

Si la función devuelve un valor, en el punto en el que se realiza la llamada, debe indicarse el nombre de una variable en el que se guarda el valor devuelto. Si no se indica el nombre de ninguna variable, JavaScript **no muestra ningún error** y el valor devuelto por la función simplemente se pierde y por tanto, no se utilizará en el resto del programa. Todas las instrucciones que se incluyen después de un **return** se ignoran y por ese motivo la instrucción **return** suele ser la última de la mayoría de funciones.

### 2.11.2. Funciones anónimas

También podemos crear funciones sin necesidad de darle un nombre. Esto es lo que se conoce como funciones anónimas.

---

```
1
2 function () {
3   console.log('Esta función no tiene nombre')
4 }
```

---

Las funciones anónimas también puede recibir parámetros.

---

```
1 (function (uno, dos, tres) {
2   console.log(uno);
3   console.log(dos);
4   console.log(tres);
5 }(1, 2, 3));
```

---

Las funciones anónimas también se pueden guardar en una variable.

---

```
1 // Función con nombre
2 function saludar(nombre)
3 {
4   console.log("Hola " + nombre);
5 }
6
7 // Llamada
8 saludar("Pepe");
9
10
11 // Función anónima guardada en la variable saludo para ser
    reutilizada
12 var saludo = function (nombre)
13 {
14   console.log("Hola " + nombre);
15 }
16
17 // Llamada
18 saludo("Pepe");
19
20
21 var resultado = function(a,b){
22   return a * b;
23 }
24
25 // Llamada
26 console.log(resultado(2,3));
```

---

### 2.11.3. Objeto Function

Cada función en JavaScript es un objeto de tipo Function y como todos los demás objetos se puede crear utilizando la instrucción new.

---

```
1 new Function ([arg1[, arg2[, ... argN]],] cuerpo-de-la-función)
```

---



```
1   var multiplicar = new Function("x", "y", "return x * y");
2
3   var laRespuesta = multiplicar(7, 6);
4   console.log(laRespuesta); // 42
5   var miEdad = 50;
6
7   if (miEdad >= 39) {
8       miEdad = multiplicar(miEdad, .5);
9   }
10  console.log(miEdad); // 25
11
12  // Función con varias sentencias
13  var multiplicar2 = new Function("x","y"," var resultado;
14  resultado = x * y; console.log(resultado)");
15  multiplicar2(2,3); //6
```

---

#### 2.11.4. Funciones flecha

La expresión de función flecha (también conocida como función flecha gruesa) dispone de una sintaxis corta comparada con la expresión de una función convencional.

Las funciones flecha siempre son anónimas.

La funciones flecha no exponen el objeto arguments.

```
1  // Sintaxis básica:
2  (param1, param2, paramN) => { statements }
3
4  (param1, param2, paramN) => expression
5  // Es como { return expression; }
6
7  // Los paréntesis son opcionales cuando solo hay un argumento:
8  singleParam => { statements }
9  singleParam => expression
10
11 // Una función sin argumentos requiere paréntesis:
12 () => { statements }
```

---

<https://desarrolloweb.com/articulos/arrow-functions-es6.html>

```
1  var materiales = [
2    'Hydrogen',
3    'Helium',
4    'Lithium',
5    'Beryllium'
6  ];
7
8  // Programación funcional.
9  console.log(materiales.map(material => material.length));
10 // Salida esperada: Array [8, 6, 7, 9]
```

---