TEMA 2

OBJETOS PREDEFINIDOS DE JAVASCRIPT

1. Cadenas de JavaScript
    - 1.1. Longitud de la cadena
    - 1.2. Caracteres especiales
    - 1.3. Rompiendo Líneas de Código Largas
    - 1.4. Las cadenas pueden ser objetos
    - 1.5. Métodos de string JavaScript
    - 1.6. Métodos de cadena y propiedades
        - 1.6.1. Longitud de la cadena
        - 1.6.2. Encontrar una cadena en una cadena
        - 1.6.3. Búsqueda de una cadena en una cadena
    - 1.7. Extracción de piezas de cadenas
        - 1.7.1. El método slice ()
        - 1.7.2. El método substring ()
        - 1.7.3. El método substr ()
    - 1.8. Sustitución de contenido de cadena
    - 1.9. Conversión a mayúsculas y minúsculas
    - 1.10. El método concat ()
    - 1.11. Extracción de caracteres de cadena
        - 1.11.1. El método charAt ()
        - 1.11.2. El método charCodeAt ()
    - 1.12. Acceso a una cadena como matriz es inseguro
    - 1.13. Convertir una cadena en una matriz
    - 1.14. Referencia de cadenas completas
2. El objeto Math de JavaScript
    - 2.1. Math.round ()
    - 2.2. Math.pow ()
    - 2.3. Math.sqrt ()
    - 2.4. Math.abs ()
    - 2.5. Math.ceil ()
    - 2.6. Math.floor ()
    - 2.7. Math.sin ()
    - 2.8. Math.cos ()
    - 2.9. Math.min () y Math.max ()
    - 2.10. Math.random ()
    - 2.11. Propiedades matemáticas (Constantes)
    - 2.12. Constructor de Math
    - 2.13. Referencia Completa de Matemáticas
3. Fechas de JavaScript
    - 3.1. Formatos de fecha de JavaScript
    - 3.2. Mostrando fechas
    - 3.3. Creación de objetos de fecha
    - 3.4. Métodos de Date
    - 3.5. Mostrando fechas
    - 3.6. Zonas horarias
    - 3.7. Métodos de fecha JavaScript
        - 3.7.1. Métodos de fecha GET
        - 3.7.2. El método getTime ()
        - 3.7.3. El método getFullYear ()
        - 3.7.4. El método getDay ()
    - 3.8. Métodos de configuración de fecha
        - 3.8.1. El método setFullYear ()

JavaScript define algunos **objetos** de forma **nativa**, por lo que pueden ser utilizados directamente por las aplicaciones sin tener que declararlos.

Cadenas de JavaScript

Las cadenas de JavaScript se utilizan para almacenar y manipular texto.

Una cadena JavaScript simplemente almacena una serie de caracteres como "John Doe".

Una cadena puede ser cualquier texto dentro de comillas. Puede usar comillas simples o dobles:

Ejemplo

```
var carname = "Volvo XC60";
var carname = 'Volvo XC60';
```

Inténtalo tú mismo "

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

Ejemplo

```
var answer = "It's alright";
var answer = "He is called 'Johnny'";
var answer = 'He is called "Johnny"';
```

Inténtalo tú mismo "

Longitud de la cadena

La longitud de una cadena se encuentra en la longitud de propiedad incorporada :

Ejemplo

```
var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
var sln = txt.length;
```

Inténtalo tú mismo "

Caracteres especiales

Debido a que las cadenas deben estar escritas entre comillas, JavaScript no interpretará correctamente esta cadena:

```
var y = "We are the so-called "Vikings" from the north."
```

La cadena será cortada a `"We are the so-called "`

La solución para evitar este problema, es utilizar el carácter \ escape .

El carácter de escape de barra invertida convierte caracteres especiales en caracteres de cadena:

Ejemplo

```
var x = 'It\'s alright';
var y = "We are the so-called \"Vikings\" from the north."
```

Inténtalo tú mismo "

El carácter de escape (\) también se puede utilizar para insertar otros caracteres especiales en una cadena.

Estos son los caracteres especiales de uso general que se pueden insertar en un texto con el signo de barra invertida:

Code	Outputs
\'	single quote
\"	double quote
\\	backslash

Cinco otros caracteres de escape son válidos en JavaScript:

Code	Outputs
\b	Backspace
\r	Carriage Return
\f	Form Feed
\t	Horizontal Tabulator
\v	Vertical Tabulator

Los 5 caracteres de escape anteriores fueron diseñados originalmente para controlar máquinas de escribir, teletipos y máquinas de fax. No tienen ningún sentido en HTML.

Rompiendo Líneas de Código Largas

Para una mejor legibilidad, a los programadores a menudo les gusta evitar líneas de código de más de 80 caracteres.

Si una declaración de JavaScript no encaja en una línea, el mejor lugar para romper es después de un operador:

Ejemplo

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

Inténtalo tú mismo "

También puede dividir una línea de código **dentro de una cadena de texto** con una sola barra diagonal inversa:

Ejemplo

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

Inténtalo tú mismo "

El método \ no es el método preferido. Puede que no tenga apoyo universal.
Algunos navegadores no permiten espacios detrás del \ carácter.

Una manera más segura de romper una cadena, es usar la adición de cadena:

Ejemplo

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

Inténtalo tú mismo "

No se puede dividir una línea de código con una barra invertida:

Ejemplo

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

Inténtalo tú mismo "

Las cadenas pueden ser objetos

Normalmente, las cadenas de JavaScript son valores primitivos, creados a partir de literales:

var firstName = "Juan";

Pero las cadenas también se pueden definir como objetos con la palabra clave new:

var firstName = new String ("John");

Ejemplo

```
var x = "John";
var y = new String("John");

// typeof x will return string
// typeof y will return object
```

Inténtalo tú mismo "

No cree cadenas como objetos. Se ralentiza la velocidad de ejecución.
La nueva palabra clave complica el código. Esto puede producir algunos resultados inesperados:

Cuando se utiliza el operador ==, las cadenas iguales son iguales:

Ejemplo

```
var x = "John";
var y = new String("John");

// (x == y) is true because x and y have equal values
```

Inténtalo tú mismo "

Cuando se utiliza el operador ===, las cadenas iguales no son iguales, porque el operador === espera igualdad tanto en el tipo como en el valor.

Ejemplo

```
var x = "John";
var y = new String("John");

// (x === y) is false because x and y have different types (string and object)
```

Inténtalo tú mismo "

O incluso peor. Los objetos no se pueden comparar:

Ejemplo

```
var x = new String("John");
var y = new String("John");

// (x == y) is false because x and y are different objects
```

Inténtalo tú mismo "

Ejemplo

```
var x = new String("John");
var y = new String("John");

// (x === y) is false because x and y are different objects
```

Inténtalo tú mismo "

Observe la diferencia entre (x == y) y (x === y).
La comparación de dos objetos JavaScript siempre devolverá false.

¡Pruebe usted mismo con ejercicios!

Ejercicio 1 » Ejercicio 2» Ejercicio 3 » Ejercicio 4»

Métodos de string JavaScript

Los métodos de cadenas le ayudan a trabajar con cadenas.

Métodos de cadena y propiedades

Los valores primitivos, como "John Doe", no pueden tener propiedades o métodos (porque no son objetos).

Pero con JavaScript, métodos y propiedades también están disponibles para valores primitivos, ya que JavaScript trata valores primitivos como objetos al ejecutar métodos y propiedades.

Longitud de la cadena

La propiedad length devuelve la longitud de una cadena:

Ejemplo

```
var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
var sln = txt.length;
```

Inténtalo tú mismo "

Encontrar una cadena en una cadena

El método indexOf () devuelve el índice de (la posición de) la primera aparición de un texto especificado en una cadena:

Ejemplo

```
var str = "Please locate where 'locate' occurs!";
var pos = str.indexOf("locate");
```

Inténtalo tú mismo "

El método lastIndexOf () devuelve el índice de la última aparición de un texto especificado en una cadena:

Ejemplo

```
var str = "Please locate where 'locate' occurs!";
var pos = str.lastIndexOf("locate");
```

Inténtalo tú mismo "

Los métodos indexOf () y lastIndexOf () devuelven -1 si no se encuentra el texto.

JavaScript cuenta las posiciones desde cero.
0 es la primera posición en una cadena, 1 es la segunda, 2 es la tercera ...

Ambos métodos aceptan un segundo parámetro como la posición inicial para la búsqueda:

Ejemplo

```
var str = "Please locate where 'locate' occurs!";
var pos = str.indexOf("locate",15);
```

Inténtalo tú mismo "

Búsqueda de una cadena en una cadena

El método search () busca una cadena para un valor especificado y devuelve la posición de la coincidencia:

Ejemplo

```
var str = "Please locate where 'locate' occurs!";
var pos = str.search("locate");
```

Inténtalo tú mismo "

¿Te diste cuenta?

Los dos métodos, indexOf () y search (), son iguales?

Aceptan los mismos argumentos (parámetros) y devuelven el mismo valor?

Los dos métodos son bastante iguales. Estas son las diferencias:

- El método search () no puede tomar un segundo argumento de posición de inicio.
- El método search () puede tomar valores de búsqueda mucho más potentes (expresiones regulares).

Aprenderá más sobre expresiones regulares en un capítulo posterior.

Extracción de piezas de cadenas

Existen 3 métodos para extraer una parte de una cadena:

- slice (comienzo, final)
- substring (inicio, fin)
- substr (comienzo, longitud)

El método slice ()

slice() extrae una parte de una cadena y devuelve la parte extraída en una nueva cadena.

El método toma 2 parámetros: el índice inicial (posición) y el índice final (posición).

Este ejemplo corta una parte de una cadena de la posición 7 a la posición 13:

Ejemplo

```
var str = "Apple, Banana, Kiwi";
var res = str.slice(7, 13);
```

El resultado de res será:

```
Banana
```

Inténtalo tú mismo "

Si un parámetro es negativo, la posición se cuenta desde el final de la cadena.

Este ejemplo corta una porción de una cadena de la posición -12 a la posición -6:

Ejemplo

```
var str = "Apple, Banana, Kiwi";
var res = str.slice(-12, -6);
```

El resultado de res será:

```
Banana
```

Inténtalo tú mismo "

Si omite el segundo parámetro, el método cortará el resto de la cadena:

Ejemplo

```
var res = str.slice(7);
```

Inténtalo tú mismo "

o, contando desde el final:

Ejemplo

```
var res = str.slice(-12);
```

Inténtalo tú mismo "

Las posiciones negativas no funcionan en Internet Explorer 8 y versiones anteriores.

El método substring ()

substring () es similar a slice ().

La diferencia es que substring () no puede aceptar índices negativos.

Ejemplo

```
var str = "Apple, Banana, Kiwi";
var res = str.substring(7, 13);
```

El resultado de res será:

```
Banana
```

Inténtalo tú mismo "

Si omite el segundo parámetro, substring () cortará el resto de la cadena.

El método substr ()

substr () es similar a slice ().

La diferencia es que el segundo parámetro especifica la longitud de la parte extraída.

Ejemplo

```
var str = "Apple, Banana, Kiwi";
var res = str.substr(7, 6);
```

El resultado de res será:

```
Banana
```

Inténtalo tú mismo "

Si el primer parámetro es negativo, la posición cuenta desde el final de la cadena.

El segundo parámetro no puede ser negativo, ya que define la longitud.

Si omite el segundo parámetro, substr () cortará el resto de la cadena.

Sustitución de contenido de cadena

El método replace () reemplaza un valor especificado por otro valor en una cadena:

Ejemplo

```
str = "Please visit Microsoft!";
var n = str.replace("Microsoft", "W3Schools");
```

Inténtalo tú mismo "

El método replace () no cambia la cadena en la que se llama. Devuelve una nueva cadena.

De forma predeterminada, la función replace () reemplaza sólo la primera coincidencia:

Ejemplo

```
str = "Please visit Microsoft and Microsoft!";
var n = str.replace("Microsoft", "W3Schools");
```

Inténtalo tú mismo "

Para reemplazar todas las coincidencias, utilice una expresión regular con un indicador / g (coincidencia global):

Ejemplo

```
str = "Please visit Microsoft and Microsoft!";
var n = str.replace(/Microsoft/g, "W3Schools");
```

Inténtalo tú mismo "

De forma predeterminada, la función replace () distingue entre mayúsculas y minúsculas. Escritura MICROSOFT (con mayúsculas) no funcionará:

Ejemplo

```
str = "Please visit Microsoft!";
var n = str.replace("MICROSOFT", "W3Schools");
```

Inténtalo tú mismo "

Para reemplazar mayúsculas y minúsculas, utilice una expresión regular con un indicador / i (insensible):

Ejemplo

```
str = "Please visit Microsoft!";
var n = str.replace(/MICROSOFT/i, "W3Schools");
```

Inténtalo tú mismo "

Aprenderá mucho más sobre las expresiones regulares en el capítulo JavaScript Expressions regulares .

Conversión a mayúsculas y minúsculas

Una cadena se convierte en mayúsculas con toUpperCase () :

Ejemplo

```
var text1 = "Hello World!";       // String
var text2 = text1.toUpperCase();  // text2 is text1 converted to upper
```

Inténtalo tú mismo "

Una cadena se convierte en minúsculas con toLowerCase () :

Ejemplo

```
var text1 = "Hello World!";       // String
var text2 = text1.toLowerCase();  // text2 is text1 converted to lower
```

Inténtalo tú mismo "

El método concat ()

concat () une dos o más cadenas:

Ejemplo

```
var text1 = "Hello";
var text2 = "World";
var text3 = text1.concat(" ", text2);
```

Inténtalo tú mismo "

El método concat () se puede utilizar en lugar del operador plus. Estas dos líneas hacen lo mismo:

Ejemplo

```
var text = "Hello" + " " + "World!";
var text = "Hello".concat(" ", "World!");
```

Todos los métodos de cadena devuelven una nueva cadena. No modifican la cadena original.
Formalmente dicho: Las cadenas son inmutables: Las cadenas no se pueden cambiar, solamente substituido.

Extracción de caracteres de cadena

Existen 2 métodos seguros para extraer caracteres de cadena:

- charAt (posición)
- charCodeAt (posición)

El método charAt ()

El método charAt () devuelve el carácter en un índice especificado (posición) en una cadena:

Ejemplo

```
var str = "HELLO WORLD";
str.charAt(0);          // returns H
```

Inténtalo tú mismo "

El método charCodeAt ()

El método charCodeAt () devuelve el unicode del carácter en un índice especificado en una cadena:

Ejemplo

```
var str = "HELLO WORLD";

str.charCodeAt(0);         // returns 72
```

Inténtalo tú mismo "

Acceso a una cadena como matriz es inseguro

Es posible que haya visto código como este, accediendo a una cadena como una matriz:

```
var str = "HELLO WORLD";

str[0];                    // returns H
```

Esto es inseguro e impredecible:

- No funciona en todos los navegadores (no en IE5, IE6, IE7)
- Hace que las cadenas se parezcan a arrays (pero no lo son)
- str [0] = "H" no da un error (pero no funciona)

Si desea leer una cadena como una matriz, convierta primero a una matriz.

Convertir una cadena en una matriz

Una cadena se puede convertir en una matriz con el método split () :

Ejemplo

```
var txt = "a,b,c,d,e";   // String
txt.split(",");          // Split on commas
txt.split(" ");          // Split on spaces
txt.split("|");          // Split on pipe
```

Inténtalo tú mismo "

Si se omite el separador, la matriz devuelta contendrá toda la cadena en el índice [0].

Si el separador es "", la matriz devuelta será una matriz de caracteres individuales:

Ejemplo

```
var txt = "Hello";       // String
txt.split("");           // Split in characters
```

Inténtalo tú mismo "

Referencia de cadenas completas

Para obtener una referencia completa, vaya a nuestra Referencia de cadenas de JavaScript completa .

La referencia contiene descripciones y ejemplos de todas las propiedades de cadena y métodos.

¡Pruebe usted mismo con ejercicios!

Ejercicio 1 » Ejercicio 2» Ejercicio 3 » Ejercicio 4» Ejercicio 5 » Ejercicio 6»

El objeto Math de JavaScript

El objeto Math de JavaScript le permite realizar tareas matemáticas en números.

Ejemplo

```
Math.PI;            // returns 3.141592653589793
```

Inténtalo tú mismo "

Math.round ()

Math.round (x) devuelve el valor de x redondeado a su entero más cercano:

Ejemplo

```
Math.round(4.7);    // returns 5
Math.round(4.4);    // returns 4
```

Inténtalo tú mismo "

Math.pow ()

Math.pow (x, y) devuelve el valor de x a la potencia de y:

Ejemplo

```
Math.pow(8, 2);      // returns 64
```

Inténtalo tú mismo "

Math.sqrt ()

Math.sqrt (x) devuelve la raíz cuadrada de x:

Ejemplo

```
Math.sqrt(64);      // returns 8
```

Inténtalo tú mismo "

Math.abs ()

Math.abs (x) devuelve el valor absoluto (positivo) de x:

Ejemplo

```
Math.abs(-4.7);     // returns 4.7
```

Inténtalo tú mismo "

Math.ceil ()

Math.ceil (x) devuelve el valor de x redondeado hasta su entero más cercano:

Ejemplo

```
Math.ceil(4.4);     // returns 5
```

Inténtalo tú mismo "

Math.floor ()

Math.floor (x) devuelve el valor de x redondeado hacia abajo a su número entero más próximo:

Ejemplo

```
Math.floor(4.7);    // returns 4
```

Inténtalo tú mismo "

Math.sin ()

Math.sin (x) devuelve el seno (un valor entre -1 y 1) del ángulo x (dado en radianes).

Si quieres usar grados en lugar de radianes, tienes que convertir grados en radianes:

Ángulo en radianes = Ángulo en grados x PI / 180.

Ejemplo

```
Math.sin(90 * Math.PI / 180);     // returns 1 (the sine of 90 degrees)
```

Inténtalo tú mismo "

Math.cos ()

Math.cos (x) devuelve el coseno (un valor entre -1 y 1) del ángulo x (dado en radianes).

Si quieres usar grados en lugar de radianes, tienes que convertir grados en radianes:

Ángulo en radianes = Ángulo en grados x PI / 180.

Ejemplo

```
Math.cos(0 * Math.PI / 180);     // returns 1 (the cos of 0 degrees)
```

Inténtalo tú mismo "

Math.min () y Math.max ()

Math.min () y Math.max () se pueden utilizar para encontrar el valor más bajo o más alto en una lista de argumentos:

Ejemplo

```
Math.min(0, 150, 30, 20, -8, -200);  // returns -200
```

Inténtalo tú mismo "

Ejemplo

```
Math.max(0, 150, 30, 20, -8, -200);  // returns 150
```

Inténtalo tú mismo "

Math.random ()

Math.random () devuelve un número aleatorio entre 0 (inclusive) y 1 (exclusivo):

Ejemplo

```
Math.random();     // returns a random number
```

Inténtalo tú mismo "

Usted aprenderá más sobre Math.random () en el próximo capítulo de este tutorial.

Propiedades matemáticas (Constantes)

JavaScript proporciona 8 constantes matemáticas a las que se puede acceder con el objeto Math:

Ejemplo

```
Math.E       // returns Euler's number
Math.PI      // returns PI
Math.SQRT2   // returns the square root of 2
Math.SQRT1_2 // returns the square root of 1/2
Math.LN2     // returns the natural logarithm of 2
Math.LN10    // returns the natural logarithm of 10
Math.LOG2E   // returns base 2 logarithm of E
Math.LOG10E  // returns base 10 logarithm of E
```

Inténtalo tú mismo "

Constructor de Math

A diferencia de otros objetos globales, el objeto Math no tiene ningún constructor. Los métodos y propiedades son estáticos.

Todos los métodos y propiedades (constantes) se pueden utilizar sin crear primero un objeto Math.

Métodos de objetos matemáticos

Method	Description
abs(x)	Returns the absolute value of x
acos(x)	Returns the arccosine of x, in radians
asin(x)	Returns the arcsine of x, in radians
atan(x)	Returns the arctangent of x as a numeric value between -PI/2 and PI/2 radians
atan2(y, x)	Returns the arctangent of the quotient of its arguments
ceil(x)	Returns the value of x rounded up to its nearest integer
cos(x)	Returns the cosine of x (x is in radians)
exp(x)	Returns the value of E^x
floor(x)	Returns the value of x rounded down to its nearest integer
log(x)	Returns the natural logarithm (base E) of x
max(x, y, z, ..., n)	Returns the number with the highest value
min(x, y, z, ..., n)	Returns the number with the lowest value
pow(x, y)	Returns the value of x to the power of y
random()	Returns a random number between 0 and 1
round(x)	Returns the value of x rounded to its nearest integer

sin(x)	Returns the sine of x (x is in radians)
sqrt(x)	Returns the square root of x
tan(x)	Returns the tangent of an angle

Referencia Completa de Matemáticas

Para una referencia completa, vaya a nuestra referencia completa del objeto matemático .

La referencia contiene descripciones y ejemplos de todas las propiedades y métodos de Math.

¡Pruebe usted mismo con ejercicios!

Ejercicio 1 » Ejercicio 2» Ejercicio 3 » Ejercicio 4»

Fechas de JavaScript

El objeto Date le permite trabajar con fechas (años, meses, días, horas, minutos, segundos y milisegundos)

Formatos de fecha de JavaScript

Una fecha JavaScript se puede escribir como una cadena:

Jue 21 Sep 2017 18:06:44 GMT + 0200 (CEST)

o como un número:

1506010004514

Fechas escritas como números, especificando el número de milisegundos desde el 1 de enero de 1970, 00:00:00.

Mostrando fechas

En este tutorial utilizamos un script para mostrar las fechas dentro de un elemento <p> con id = "demo":

Ejemplo

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

<script>
document.getElementById("demo").innerHTML = Date();
</script>
```

Inténtalo tú mismo "

El script anterior dice: asignar el valor de Date () al contenido (innerHTML) del elemento con id = "demo".

Usted aprenderá a mostrar una fecha, en un formato más legible, al final de esta página.

Creación de objetos de fecha

El objeto Date nos permite trabajar con fechas.

Una fecha consiste en un año, un mes, un día, una hora, un minuto, un segundo y milisegundos.

Los objetos de fecha se crean con el nuevo constructor Date () .

Hay 4 maneras de iniciar una fecha:

```
new Date()
new Date(milliseconds)
new Date(dateString)
new Date(year, month, day, hours, minutes, seconds, milliseconds)
```

Con la nueva Date (), se crea un nuevo objeto de fecha con la fecha y la hora actuales :

Ejemplo

```
<script>
var d = new Date();
document.getElementById("demo").innerHTML = d;
</script>
```

Inténtalo tú mismo "

Con la nueva Fecha (cadena de fecha), se crea un nuevo objeto de fecha a partir de la fecha y hora especificadas :

Ejemplo

```
<script>
var d = new Date("October 13, 2014 11:13:00");
document.getElementById("demo").innerHTML = d;
</script>
```

Inténtalo tú mismo "

Las cadenas de fecha válidas (formatos de fecha) se describen en el siguiente capítulo.

Utilizando nueva fecha (número), crea un nuevo objeto de fecha como hora cero más el número .

La hora cero es 01 enero 1970 00:00:00 UTC. El número se especifica en milisegundos:

Ejemplo

```
<script>
var d = new Date(86400000);
document.getElementById("demo").innerHTML = d;
</script>
```

Inténtalo tú mismo "

Las fechas de JavaScript se calculan en milisegundos desde el 01 enero, 1970 00:00:00 Hora Mundial (UTC). Un día contiene 86.400.000 milisegundos.

Utilizando new Date (7 números), crea un nuevo objeto de fecha con la fecha y la hora especificadas :

Los 7 números especifican el año, mes, día, hora, minuto, segundo y milisegundo, en ese orden:

Ejemplo

```
<script>
var d = new Date(99, 5, 24, 11, 33, 30, 0);
document.getElementById("demo").innerHTML = d;
</script>
```

Inténtalo tú mismo "

Las variantes del ejemplo anterior permiten omitir cualquiera de los 4 últimos parámetros:

Ejemplo

```
<script>
var d = new Date(99, 5, 24);
document.getElementById("demo").innerHTML = d;
</script>
```

Inténtalo tú mismo "

JavaScript cuenta meses de 0 a 11. Enero es 0. Diciembre es 11.

Métodos de Date

Cuando se crea un objeto Date, varios métodos permiten operar en él.

Los métodos de fecha permiten obtener y configurar el año, mes, día, hora, minuto, segundo y milisegundo de los objetos, utilizando la hora local o la hora UTC (universal o GMT).

Los métodos de fechas se tratan en un capítulo posterior.

Mostrando fechas

Cuando se muestra un objeto de fecha en HTML, se convierte automáticamente en una cadena, con el método toString () .

Ejemplo

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

```
<script>
d = new Date();
document.getElementById("demo").innerHTML = d;
</script>
```

Es lo mismo que:

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

<script>
d = new Date();
document.getElementById("demo").innerHTML = d.toString();
</script>
```

Inténtalo tú mismo "

El método toUTCString () convierte una fecha en una cadena UTC (un estándar de visualización de fecha).

Ejemplo

```
<script>
var d = new Date();
document.getElementById("demo").innerHTML = d.toUTCString();
</script>
```

Inténtalo tú mismo "

El método toDateString () convierte una fecha en un formato más legible:

Ejemplo

```
<script>
var d = new Date();
document.getElementById("demo").innerHTML = d.toDateString();
</script>
```

Inténtalo tú mismo "

> Los objetos de fecha son estáticos. El tiempo de la computadora está marcando, pero los objetos de fecha, una vez creados, no lo son.

Zonas horarias

Al establecer una fecha, sin especificar la zona horaria, JavaScript utilizará la zona horaria del navegador.

Al obtener una fecha, sin especificar la zona horaria, el resultado se convierte en la zona horaria del navegador.

En otras palabras: Si se crea una fecha / hora en GMT (hora media de Greenwich), la fecha y la hora se convertirán en CDT (hora central de EE.UU.) si un usuario navega desde el centro de Estados Unidos.

> Lea más sobre las zonas horarias en los próximos capítulos.

¡Pruebe usted mismo con ejercicios!

Ejercicio 1 » Ejercicio 2» Ejercicio 3 »

Métodos de fecha JavaScript

Los métodos de fecha le permiten obtener y establecer los valores de fecha (años, meses, días, horas, minutos, segundos, milisegundos)

Métodos de fecha GET

Los métodos get se utilizan para obtener una parte de una fecha. Aquí están los más comunes (alfabéticamente):

Method	Description
getDate()	Get the day as a number (1-31)
getDay()	Get the weekday as a number (0-6)
getFullYear()	Get the four digit year (yyyy)
getHours()	Get the hour (0-23)
getMilliseconds()	Get the milliseconds (0-999)
getMinutes()	Get the minutes (0-59)
getMonth()	Get the month (0-11)
getSeconds()	Get the seconds (0-59)
getTime()	Get the time (milliseconds since January 1, 1970)

El método getTime ()

getTime () devuelve el número de milisegundos desde el 1 de enero de 1970:

Ejemplo

```
<script>
var d = new Date();
document.getElementById("demo").innerHTML = d.getTime();
</script>
```

Inténtalo tú mismo "

El método getFullYear ()

getFullYear () devuelve el año de una fecha como un número de cuatro dígitos:

Ejemplo

```
<script>
var d = new Date();
document.getElementById("demo").innerHTML = d.getFullYear();
</script>
```

Inténtalo tú mismo "

El método getDay ()

getDay () devuelve el día de la semana como un número (0-6):

Ejemplo

```
<script>
var d = new Date();
document.getElementById("demo").innerHTML = d.getDay();
</script>
```

Inténtalo tú mismo "

En JavaScript, el primer día de la semana (0) significa "Domingo", incluso si algunos países en el mundo consideran que el primer día de la semana es "Lunes"

Puede utilizar una matriz de nombres y getDay () para devolver el día de la semana como un nombre:

Ejemplo

```
<script>
var d = new Date();
var days =
["Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday
"];
document.getElementById("demo").innerHTML = days[d.getDay()];
</script>
```

Inténtalo tú mismo "

Métodos de configuración de fecha

Los métodos de ajuste se utilizan para establecer una parte de una fecha. Aquí están los más comunes (alfabéticamente):

Method	Description
setDate()	Set the day as a number (1-31)
setFullYear()	Set the year (optionally month and day)
setHours()	Set the hour (0-23)
setMilliseconds()	Set the milliseconds (0-999)
setMinutes()	Set the minutes (0-59)
setMonth()	Set the month (0-11)
setSeconds()	Set the seconds (0-59)
setTime()	Set the time (milliseconds since January 1, 1970)

El método setFullYear ()

setFullYear () establece un objeto de fecha en una fecha específica. En este ejemplo, al 14 de enero de 2020:

Ejemplo

```
<script>
var d = new Date();
d.setFullYear(2020, 0, 14);
document.getElementById("demo").innerHTML = d;
</script>
```

Inténtalo tú mismo "

El método setDate ()

setDate () establece el día del mes (1-31):

Ejemplo

```
<script>
var d = new Date();
d.setDate(20);
document.getElementById("demo").innerHTML = d;
</script>
```

Inténtalo tú mismo "

El método setDate () también se puede usar para agregar días a una fecha:

Ejemplo

```
<script>
var d = new Date();
d.setDate(d.getDate() + 50);
document.getElementById("demo").innerHTML = d;
</script>
```

Inténtalo tú mismo "

Si agrega días, cambia el mes o el año, los cambios son manejados automáticamente por el objeto Fecha.

Fecha de entrada - Parsing Dates

Si tiene una cadena de fecha válida, puede utilizar el método Date.parse () para convertirlo en milisegundos.

Date.parse () devuelve el número de milisegundos entre la fecha y el 1 de enero de 1970:

Ejemplo

```
<script>
var msec = Date.parse("March 21, 2012");
document.getElementById("demo").innerHTML = msec;
</script>
```

Inténtalo tú mismo "

A continuación, puede utilizar el número de milisegundos para convertirlo en un objeto de fecha :

Ejemplo

```
<script>
var msec = Date.parse("March 21, 2012");
var d = new Date(msec);
document.getElementById("demo").innerHTML = d;
</script>
```

Inténtalo tú mismo "

Comparar fechas

Las fechas pueden compararse fácilmente.

El siguiente ejemplo compara la fecha de hoy con el 14 de enero de 2100:

Ejemplo

```
var today, someday, text;
today = new Date();
someday = new Date();
someday.setFullYear(2100, 0, 14);

if (someday > today) {
    text = "Today is before January 14, 2100.";
} else {
    text = "Today is after January 14, 2100.";
}
document.getElementById("demo").innerHTML = text;
```

Inténtalo tú mismo "

JavaScript cuenta meses de 0 a 11. Enero es 0. Diciembre es 11.

Métodos de la fecha UTC

Métodos de fecha UTC se utilizan para trabajar las fechas UTC (fechas de zona horaria universal):

Method	Description
getUTCDate()	Same as getDate(), but returns the UTC date
getUTCDay()	Same as getDay(), but returns the UTC day
getUTCFullYear()	Same as getFullYear(), but returns the UTC year
getUTCHours()	Same as getHours(), but returns the UTC hour
getUTCMilliseconds()	Same as getMilliseconds(), but returns the UTC milliseconds
getUTCMinutes()	Same as getMinutes(), but returns the UTC minutes
getUTCMonth()	Same as getMonth(), but returns the UTC month
getUTCSeconds()	Same as getSeconds(), but returns the UTC seconds

---

Referencia completa de fecha de JavaScript

Para una referencia completa, vaya a nuestra Referencia completa de fecha de JavaScript .

La referencia contiene descripciones y ejemplos de todas las propiedades y métodos Date.

Números de JavaScript

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

Ejemplo

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

Inténtalo tú mismo "

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

Ejemplo

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

Inténtalo tú mismo "

Los números JavaScript siempre son de 64 bits punto flotante

A diferencia de muchos otros lenguajes de programación, JavaScript no define diferentes tipos de números, como enteros, corto, largo, punto flotante, etc.

Los números JavaScript se almacenan siempre como números de coma flotante de precisión doble, siguiendo el estándar internacional IEEE 754.

Este formato almacena números en 64 bits, donde el número (la fracción) se almacena en los bits 0 a 51, el exponente en los bits 52 a 62 y el signo en el bit 63:

Valor (también conocido como Fracción / Mantisa)	Exponente	Firmar
52 bits (0 - 51)	11 bits (52 - 62)	1 bit (63)

Precisión

Enteros (números sin una notación de período o exponente) son exactos hasta 15 dígitos.

Ejemplo

```
var x = 999999999999999;   // x will be 999999999999999
var y = 9999999999999999;  // y will be 10000000000000000
```

Inténtalo tú mismo "

El número máximo de decimales es 17, pero la aritmética de punto flotante no siempre es 100% exacta:

Ejemplo

```
var x = 0.2 + 0.1;         // x will be 0.30000000000000004
```

Inténtalo tú mismo "

Para resolver el problema anterior, ayuda a multiplicar y dividir:

Ejemplo

```
var x = (0.2 * 10 + 0.1 * 10) / 10;       // x will be 0.3
```

Inténtalo tú mismo "

Adición de números y cadenas

ADVERTENCIA !!

JavaScript utiliza el operador + para la adición y la concatenación.

Se agregan números. Las cadenas están concatenadas.

Si agrega dos números, el resultado será un número:

Ejemplo

```
var x = 10;
var y = 20;
var z = x + y;          // z will be 30 (a number)
```

Inténtalo tú mismo "

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

Ejemplo

```
var x = "10";
var y = "20";
var z = x + y;          // z will be 1020 (a string)
```

Inténtalo tú mismo "

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

Ejemplo

```
var x = 10;
var y = "20";
var z = x + y;          // z will be 1020 (a string)
```

Inténtalo tú mismo "

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

Ejemplo

```
var x = "10";
var y = 20;
var z = x + y;          // z will be 1020 (a string)
```

Inténtalo tú mismo "

Un error común es esperar que este resultado sea 30:

Ejemplo

```
var x = 10;
var y = 20;
var z = "The result is: " + x + y;
```

Inténtalo tú mismo "

Un error común es esperar que este resultado sea 102030:

Ejemplo

```
var x = 10;
var y = 20;
var z = "30";
var result = x + y + z;
```

Inténtalo tú mismo "

El compilador JavaScript funciona de izquierda a derecha.

Los primeros 10 + 20 se añaden porque x e y son ambos números.

Entonces 30 + "30" está concatenado porque z es una cadena.

Cadenas numéricas

Las cadenas de JavaScript pueden tener contenido numérico:

```
var x = 100;        // x is a number

var y = "100";      // y is a string
```

JavaScript intentará convertir cadenas a números en todas las operaciones numéricas:

Esto funcionará:

```
var x = "100";
var y = "10";
var z = x / y;      // z will be 10
```

Inténtalo tú mismo "

Esto también funcionará:

```
var x = "100";
var y = "10";
var z = x * y;      // z will be 1000
```

Inténtalo tú mismo "

Y esto funcionará:

```
var x = "100";
var y = "10";
var z = x - y;      // z will be 90
```

Inténtalo tú mismo "

Pero esto no funcionará:

```
var x = "100";
var y = "10";
var z = x + y;      // z will not be 110 (It will be 10010)
```

Inténtalo tú mismo "

En el último ejemplo, JavaScript utiliza el operador + para concatenar las cadenas.

NaN - No es un número

NaN es una palabra reservada JavaScript que indica que un número no es un número legal.

Tratar de hacer aritmética con una cadena no numérica resultará en NaN (Not a Number):

Ejemplo

```
var x = 100 / "Apple";  // x will be NaN (Not a Number)
```

Inténtalo tú mismo "

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

Ejemplo

```
var x = 100 / "10";     // x will be 10
```

Inténtalo tú mismo "

Puede utilizar la función JavaScript global isNaN () para averiguar si un valor es un número:

Ejemplo

```
var x = 100 / "Apple";
isNaN(x);               // returns true because x is Not a Number
```

Inténtalo tú mismo "

Cuidado con NaN. Si utiliza NaN en una operación matemática, el resultado también será NaN:

Ejemplo

```
var x = NaN;
var y = 5;
var z = x + y;          // z will be NaN
```

Inténtalo tú mismo "

O el resultado podría ser una concatenación:

Ejemplo

```
var x = NaN;
var y = "5";
var z = x + y;        // z will be NaN5
```

Inténtalo tú mismo "

NaN es un número: typeof NaN devuelve el número:

Ejemplo

```
typeof NaN;           // returns "number"
```

Inténtalo tú mismo "

---

infinito

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

Ejemplo

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

Inténtalo tú mismo "

División por 0 (cero) también genera Infinity:

Ejemplo

```
var x =  2 / 0;         // x will be Infinity
var y = -2 / 0;         // y will be -Infinity
```

Inténtalo tú mismo "

Infinity es un número: typeof Infinity devuelve el número.

Ejemplo

```
typeof Infinity;        // returns "number"
```

Inténtalo tú mismo "

Hexadecimal

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

Ejemplo

```
var x = 0xFF;           // x will be 255
```

Inténtalo tú mismo "

Nunca escriba un número con un cero inicial (como 07).
Algunas versiones de JavaScript interpretan los números como octales si se escriben con un cero inicial.

De forma predeterminada, JavaScript muestra números como 10 décimas de base.

Pero puede utilizar el método toString () para dar salida a los números como base 16 (hex), base 8 (octal) o base 2 (binario).

Ejemplo

```
var myNumber = 128;
myNumber.toString(16);  // returns 80
myNumber.toString(8);   // returns 200
myNumber.toString(2);   // returns 10000000
```

Inténtalo tú mismo "

Los números pueden ser objetos

Normalmente, los números de JavaScript son valores primitivos creados a partir de literales:

var x = 123;

Pero los números también pueden definirse como objetos con la palabra clave new:

var y = new Number (123);

Ejemplo

```
var x = 123;
var y = new Number(123);

// typeof x returns number
// typeof y returns object
```

Inténtalo tú mismo "

No cree objetos numéricos. Se ralentiza la velocidad de ejecución.
La nueva palabra clave complica el código. Esto puede producir algunos resultados inesperados:

Cuando se utiliza el operador ==, los números iguales son iguales:

Ejemplo

```
var x = 500;
var y = new Number(500);

// (x == y) is true because x and y have equal values
```

Inténtalo tú mismo "

Cuando se utiliza el operador ===, los números iguales no son iguales, porque el operador === espera igualdad tanto en el tipo como en el valor.

Ejemplo

```
var x = 500;
var y = new Number(500);

// (x === y) is false because x and y have different types
```

Inténtalo tú mismo "

O incluso peor. Los objetos no se pueden comparar:

Ejemplo

```
var x = new Number(500);
var y = new Number(500);

// (x == y) is false because objects cannot be compared
```

Inténtalo tú mismo "

Observe la diferencia entre (x == y) y (x === y).
La comparación de dos objetos JavaScript siempre devolverá false.

¡Pruebe usted mismo con ejercicios!

Ejercicio 1 » Ejercicio 2» Ejercicio 3 » Ejercicio 4»

Interacción de los objetos con el navegador

JavaScript Window Navigator

El objeto window.navigator contiene información sobre el navegador del visitante.

Window navigator

El objeto window.navigator se puede escribir sin el prefijo window.

Algunos ejemplos:

- navigator.appName
- navigator.appCodeName
- navigator.platform

Cookies del navegador

La propiedad cookieEnabled devuelve true si las cookies están habilitadas, de lo contrario false:

Ejemplo

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

<script>
document.getElementById("demo").innerHTML =
"cookiesEnabled is " + navigator.cookieEnabled;
</script>
```

Inténtalo tú mismo "

Nombre de la aplicación del navegador

La propiedad appName devuelve el nombre de la aplicación del navegador:

Ejemplo

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

<script>
document.getElementById("demo").innerHTML =
"navigator.appName is " + navigator.appName;
</script>
```

Inténtalo tú mismo "

Curiosamente, "Netscape" es el nombre de la aplicación para IE11, Chrome, Firefox y Safari.

Nombre del código de aplicación del navegador

La propiedad appCodeName devuelve el nombre de código de la aplicación del explorador:

Ejemplo

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

<script>
document.getElementById("demo").innerHTML =
"navigator.appCodeName is " + navigator.appCodeName;
</script>
```

Inténtalo tú mismo "

"Mozilla" es el nombre de código de la aplicación para Chrome, Firefox, IE, Safari y Opera.

El motor del navegador

La propiedad product devuelve el nombre del producto del motor del navegador:

Ejemplo

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

<script>
document.getElementById("demo").innerHTML =
"navigator.product is " + navigator.product;
</script>
```

Inténtalo tú mismo "

No confíe en esto. La mayoría de los navegadores devuelven "Gecko" como nombre de producto !!

La versión del navegador

La propiedad appVersion devuelve información sobre la versión del navegador:

Ejemplo

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

<script>
document.getElementById("demo").innerHTML = navigator.appVersion;
</script>
```

Inténtalo tú mismo "

El agente del navegador

La propiedad userAgent devuelve el encabezado de agente de usuario enviado por el explorador al servidor:

Ejemplo

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

<script>
document.getElementById("demo").innerHTML = navigator.userAgent;
</script>
```

Inténtalo tú mismo "

Advertencia !!!

La información del objeto de navegación a menudo puede ser engañosa y no debe utilizarse para detectar versiones de navegador porque:

- Diferentes navegadores pueden usar el mismo nombre
- Los datos del navegador pueden ser cambiados por el propietario del navegador
- Algunos navegadores se identifican erróneamente para evitar las pruebas del sitio
- Los navegadores no pueden informar de nuevos sistemas operativos, publicados más tarde que el navegador

---

La plataforma del navegador

La propiedad platform devuelve la plataforma del explorador (sistema operativo):

Ejemplo

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

<script>
document.getElementById("demo").innerHTML = navigator.platform;
</script>
```

Inténtalo tú mismo "

El lenguaje del navegador

La propiedad language devuelve el idioma del navegador:

Ejemplo

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

<script>
document.getElementById("demo").innerHTML = navigator.language;
</script>
```

Inténtalo tú mismo "

¿Está el navegador en línea?

La propiedad onLine devuelve true si el navegador está en línea:

Ejemplo

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

<script>
document.getElementById("demo").innerHTML = navigator.onLine;
</script>
```

Inténtalo tú mismo "

¿Está habilitado Java?

El método javaEnabled () devuelve true si Java está habilitado:

Ejemplo

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

<script>
document.getElementById("demo").innerHTML = navigator.javaEnabled();
</script>
```

Inténtalo tú mismo "

JavaScript Window Screen

El objeto window.screen contiene información sobre la pantalla del usuario.

Pantalla de la ventana

El objeto window.screen puede escribirse sin el prefijo window.

Propiedades:

- screen.width
- screen.height
- screen.availWidth
- screen.availHeight
- screen.colorDepth
- screen.pixelDepth

Ancho de pantalla de la ventana

La propiedad screen.width devuelve el ancho de la pantalla del visitante en píxeles.

Ejemplo

Muestra el ancho de la pantalla en píxeles:

```
document.getElementById("demo").innerHTML =
"Screen Width: " + screen.width;
```

El resultado será:

```
Screen Width: 1366
```

Inténtalo tú mismo "

Altura de la pantalla de la ventana

La propiedad screen.height devuelve la altura de la pantalla del visitante en píxeles.

Ejemplo

Muestra la altura de la pantalla en píxeles:

```
document.getElementById("demo").innerHTML =
"Screen Height: " + screen.height;
```

El resultado será:

```
Screen Height: 768
```

Inténtalo tú mismo "

Pantalla de la ventana Anchura disponible

La propiedad screen.availWidth devuelve el ancho de la pantalla del visitante, en píxeles, menos las características de la interfaz como la barra de tareas de Windows.

Ejemplo

Muestra el ancho disponible de la pantalla en píxeles:

```
document.getElementById("demo").innerHTML =
"Available Screen Width: " + screen.availWidth;
```

El resultado será:

```
Available Screen Width: 1366
```

Inténtalo tú mismo "

Pantalla de ventana Altura disponible

La propiedad screen.availHeight devuelve la altura de la pantalla del visitante, en píxeles, menos características de interfaz como la barra de tareas de Windows.

Ejemplo

Muestra la altura disponible de la pantalla en píxeles:

```
document.getElementById("demo").innerHTML =
"Available Screen Height: " + screen.availHeight;
```

El resultado será:

```
Available Screen Height: 768
```

Inténtalo tú mismo "

Profundidad de color de la pantalla de la ventana

La propiedad screen.colorDepth devuelve el número de bits utilizados para mostrar un color.

Todas las computadoras modernas utilizan 24 bits o hardware de 32 bits para la resolución de color:

- 24 bits = 16.777.216 diferentes "Colores Verdaderos"
- 32 bits = 4.294.967.296 diferentes "colores profundos"

Las computadoras más antiguas utilizan 16 bits: 65.536 diferentes "High Colors" de resolución.

Las computadoras muy viejas y los viejos teléfonos celulares usaban 8 bits: 256 "colores VGA" diferentes.

Ejemplo

Muestra la profundidad de color de la pantalla en bits:

```
document.getElementById("demo").innerHTML =
"Screen Color Depth: " + screen.colorDepth;
```

El resultado será:

```
Screen Color Depth: 24
```

Inténtalo tú mismo "

Los valores #rrggbb (rgb) utilizados en HTML representan "True Colors" (16.777.216 colores diferentes)

Profundidad de píxel de pantalla de ventana

La propiedad screen.pixelDepth devuelve la profundidad de píxel de la pantalla.

Ejemplo

Muestra la profundidad de píxel de la pantalla en bits:

```
document.getElementById("demo").innerHTML =
"Screen Pixel Depth: " + screen.pixelDepth;
```

El resultado será:

```
Screen Pixel Depth: 24
```

Inténtalo tú mismo "

Para las computadoras modernas, la profundidad de color y la profundidad de píxeles son iguales.

El objeto window

El objeto de window es compatible con todos los navegadores. Representa la ventana del navegador.

Todos los objetos, funciones y variables globales de JavaScript se convierten automáticamente en miembros del objeto de window..

Las variables globales son propiedades del objeto window.

Las funciones globales son métodos del objeto window.

Incluso el objeto document (del HTML DOM) es una propiedad del objeto window:

```
window.document.getElementById("header");
```

es lo mismo que:

```
document.getElementById("header");
```

Tamaño de ventana

Se pueden usar dos propiedades para determinar el tamaño de la ventana del navegador.

Ambas propiedades devuelven los tamaños en píxeles:

- window.innerHeight - la altura interna de la ventana del navegador (en píxeles)
- window.innerWidth - el ancho interno de la ventana del navegador (en píxeles)

La ventana del navegador (la ventana de visualización del navegador) NO incluye barras de herramientas ni barras de desplazamiento.

Para Internet Explorer 8, 7, 6, 5:

- document.documentElement.clientHeight
- document.documentElement.clientWidth
- o
- document.body.clientHeight
- document.body.clientWidth

Una solución práctica de JavaScript (que cubre todos los navegadores):

Ejemplo

```
var w = window.innerWidth || document.documentElement.clientWidth
|| document.body.clientWidth;

var h = window.innerHeight
|| document.documentElement.clientHeight
|| document.body.clientHeight;
```

Inténtalo tú mismo "

El ejemplo muestra la altura y el ancho de la ventana del navegador: (NO incluye barras de herramientas / barras de desplazamiento)

Otros métodos de ventana

Algunos otros métodos:

- window.open () - abre una nueva ventana
- window.close () - cierre la ventana actual
- window.moveTo () -move la ventana actual
- window.resizeTo () -resize la ventana actual

Un script no creará nunca la ventana principal de un navegador. Es el usuario, quien realiza esa tarea abriendo una URL en el navegador o un archivo desde el menú de abrir. Pero sin embargo, un script que esté ejecutándose en una de las ventanas principales del navegador, si que podrá crear o abrir nuevas sub-ventanas.

El método que genera una nueva ventana es window.open(). Este método contiene hasta tres parámetros, que definen las características de la nueva ventana: la URL del documento a abrir, el nombre de esa ventana y su apariencia física (tamaño, color,etc.).

Por ejemplo, si consideramos la siguiente instrucción que abre una nueva ventana de un tamaño determinado y con el contenido de un documento HTML:

```
var subVentana=window.open("nueva.html","nueva","height=800,width=600");
```

Lo importante de esa instrucción, es la asignación que hemos hecho en la variable subVentana. De esta forma podremos a lo largo de nuestro código, referenciar a la nueva ventana desde el script original de la ventana principal. Por ejemplo, si quisiéramos cerrar la nueva ventana desde nuestro script, simplemente tendríamos que hacer: subVentana.close();

Aquí sí que es necesario especificar subVentana, ya que si escribiéramos window.close(), self.close() o close() estaríamos intentando cerrar nuestra propia ventana (previa confirmación), pero no la subVentana que creamos en los pasos anteriores.

Véase el siguiente ejemplo que permite abrir y cerrar una sub-ventana:

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="content-type" content="text/html;charset=utf-8">
<title>Apertura y Cierre de Ventanas</title>
<script type="text/javascript">
function inicializar(){
document.getElementById("crear-ventana").onclick=crearNueva;
document.getElementById("cerrar-ventana").onclick=cerrarNueva;
}
var nuevaVentana;
function crearNueva(){
nuevaVentana = window.open("http://www.google.es","","height=400,width=800");
}
function cerrarNueva(){
if (nuevaVentana){
nuevaVentana.close(); nuevaVentana = null;
}
}
</script>
</head>
<body onLoad="inicializar()">
<h1>Abrimos y cerramos ventanas</h1>
<form>
<p> <input type="button" id="crear-ventana" value="Crear Nueva Ventana">
```

<input type="button" id="cerrar-ventana" value="Cerrar Nueva Ventana"> </p>

</form>

</body>

</html>

El objeto **window** creado por el navegador se encuentra en la raíz de la jerarquía que describe el contenido del documento HTML. A este objeto se le asocian propiedades que permiten acceder al contenido del documento así como a los métodos que permiten crear o destruir ventanas cliente y ventanas de diálogo.

Al igual que el objeto *document*, los manejadores de eventos asociados, son los mismos que están asignados a la marca *BODY*, es decir, **onLoad** y **onUnLoad**.

Las propiedades de *window*.

- **defaultStatus**. Mensaje visualizado en el parte inferior de la ventana.
- **frames**. Matriz con particiones de la ventana. El campo *frames.length* representa el número de frames que se encuentran en una ventada.
- **parent**. Ascendiente de una partición tipo FRAME. Forma la ventana padre, es decir, la que contiene la marca FRAMESET, a partir de esta, se obtiene todas las subventanas o FRAMEs.
- **self**. Hace referencia a la ventana actual.
- **status**. Mensaje que ilustra el estado del cliente.
- **top**. Raíz de la jerarquía de objetos definida por el cliente.
- **window**. Se refiere a la ventana actual.

Los métodos de *window*.

- **alert("mensaje")**. Crea una ventana de diálogo en la que se muestra el mensaje.
  Hacer doble click:
- **close()**. Destruye la ventana del cliente actual.
- **confirm("mensaje")**. Crea una ventana de confirmación *(OK/Cancel)* en la que se visualiza la cadena de caracteres *"mensaje"*. El valor devuelto es **true** si el usuario elige la opción OK y **false** en caso contrario.

- **prompt("mensaje","texto")**. Crea una ventana de díálogo en la que se visualiza la cadena de caracteres *"mensaje"* y permite la edición de una cadena de caracteres. El parámetro *"texto"* representa el valor predeterminado. Esta ventana, al igual que confirm, posee dos botones

aceptar y cancelar. Si se pulsa aceptar este método devolverá el texto insertado por el usuario o el que se daba por defecto si el usuario no ha introducido nada, mientras que si se pulsa cancelar dicho método devolverá **null**.

- **setTimeout(expresión, msec)**. Este método evalúa la expresión pasada como argumento después de que el número de milisegundos *msec* haya pasado.
- **clearTimeout(timeoutID)**. Anula la cuenta a atrás inicializada por el método setTimeout e identifica la variable *timeoutID*.
- **open("URL","WindowName","Características")**. Crea una nueva ventana cliente, le asocia el nombre *WindowName* y accede al *URL* indicado, si éste campo se pasa "en blanco" obtendremos una ventana vacía.
  La *Características* son un conjunto de parámetros que describen las propiedades de la ventana. En la siguiente tabla podemos ver cada uno de estos parámetros con su significado.

Parámetro	Descripción
toolbar	Crea una barra de herramientas con botones: *back, forward, ...*
resizable	El usuario puede modificar el tamaño de la ventana.
width	Especifica la anchura de la ventana en pixeles.
height	Especifica la anchura de la ventana en pixeles.
copyhistory	Asigna a la misma ventana el mismo histórico.
directories	Crea la barra de botones estándar del navegador
location	Crea una campo de entrada location *(especifica URL del documento)*..
status	Crea la barra de estado en la parte inferior de la ventana.
scrollbars	Crea las barras de scroll horizontal y vertical cuando sea necesario.
menubar	Crea la barra de menú en la parte superior de la ventana.

A los parámetros *height y width* se les asigna el número de pixels, mientras que al resto se les asigna *yes, 1 o true* para aceptar el parámetro y *no , 0 o false* para denegarlo.

Para ver un ejemplo sobre los métodos **open** y **close** ir al apartado de la creación de ventanas del capítulo 8.
A continuación veremos un ejemplo en el que utilizaremos los métodos del objeto windows *setTimeout* y *clearTimeout*. Además de trabajar con la propiedad *status*.

Historial de la ventana JavaScript

El objeto window.history contiene el historial del navegador.

Historia de la ventana

El objeto window.history se puede escribir sin el prefijo de ventana.

Para proteger la privacidad de los usuarios, hay limitaciones en cómo JavaScript puede acceder a este objeto.

Algunos métodos:

- history.back () - lo mismo que hacer clic en el navegador
- history.forward () - lo mismo que hacer clic en el navegador

---

Window History Back

El método history.back () carga la URL anterior en la lista del historial.

Esto es lo mismo que hacer clic en el botón Atrás en el navegador.

Ejemplo

Crear un botón de retroceso en una página:

```
<html>
<head>
<script>
function goBack() {
    window.history.back()
}
</script>
</head>
<body>

<input type="button" value="Back" onclick="goBack()">

</body>
</html>
```

Historia de la ventana

El método history forward () carga la siguiente URL en la lista del historial.

Esto es lo mismo que hacer clic en el botón Adelante en el navegador.

Ejemplo

Crear un botón de avance en una página:

```
<html>
<head>
<script>
function goForward() {
    window.history.forward()
}
</script>
</head>
<body>

<input type="button" value="Forward" onclick="goForward()">

</body>
</html>
```

JavaScript Window Location

El objeto window.location se puede utilizar para obtener la dirección de página actual (URL) y para redirigir el navegador a una nueva página.

Window Location

El objeto window.location se puede escribir sin el prefijo de ventana.

Algunos ejemplos:

- window.location.href devuelve el href (URL) de la página actual
- window.location.hostname devuelve el nombre de dominio del host web
- window.location.pathname devuelve la ruta y el nombre de archivo de la página actual
- window.location.protocol devuelve el protocolo web utilizado (http: o https :)
- window.location.assign carga un nuevo documento

Window Location Href

La propiedad window.location.href devuelve la URL de la página actual.

Ejemplo

Mostrar el href (URL) de la página actual:

```
document.getElementById("demo").innerHTML =
"Page location is " + window.location.href;
```

El resultado es:

```
Page location is https://www.w3schools.com/js/js_window_location.asp
```

Inténtalo tú mismo "

Window Location Hostname

La propiedad window.location.hostname devuelve el nombre del host de Internet (de la página actual).

Ejemplo

Mostrar el nombre del host:

```
document.getElementById("demo").innerHTML =
"Page hostname is " + window.location.hostname;
```

El resultado es:

```
Page hostname is www.w3schools.com
```

Inténtalo tú mismo "

Window Location Pathname

La propiedad window.location.pathname devuelve el nombre de ruta de la página actual.

Ejemplo

Mostrar el nombre de ruta de la URL actual:

```
document.getElementById("demo").innerHTML =
"Page path is " + window.location.pathname;
```

El resultado es:

```
/js/js_window_location.asp
```

Inténtalo tú mismo "

Window Location Protocol

La propiedad window.location.protocol devuelve el protocolo web de la página.

Ejemplo

Mostrar el protocolo web:

```
document.getElementById("demo").innerHTML =
"Page protocol is " + window.location.protocol;
```

El resultado es:

```
Page protocol is https:
```

Inténtalo tú mismo "

Window Location Port

La propiedad window.location.port devuelve el número del puerto de host de Internet (de la página actual).

Ejemplo

Mostrar el nombre del host:

```
document.getElementById("demo").innerHTML =
"Port number is " + window.location.port;
```

El resultado es:

```
Port name is
```

Inténtalo tú mismo "

> La mayoría de los navegadores no mostrarán los números de puerto predeterminados (80 para http y 443 para https)

Window Location Assign

El método window.location.assign () carga un nuevo documento.

Ejemplo

Cargar un nuevo documento:

```
<html>
<head>
<script>
function newDoc() {
    window.location.assign("https://www.w3schools.com")
}
</script>
</head>
<body>

<input type="button" value="Load new document" onclick="newDoc()">

</body>
</html>
```

Inténtalo tú mismo "

Generación de elementos HTML desde código JavaScript

Uno de los principales objetivos de JavaScript en el desarrollo web en la parte del cliente es convertir un documento HTML estático en una aplicación web dinámica. Por ejemplo, es muy común que los scripts detecten el tipo y la versión del navegador que estamos utilizando y, en base a esto escribir las etiquetas adecuadas para cada navegador. De igual modo, se suelen utilizar los valores de determinadas variables para ejecutar instrucciones que creen nuevas ventanas con contenido propio, en lugar de mostrarlo en la ventana actual. Cada ventana de un navegador presenta un documento HTML y es representada por un objeto Window que contiene un subobjeto Document. El objeto Document contiene a su vez una serie de objetos que representan todo el contenido del documento HTML, como por ejemplo el texto, las imágenes, los enlaces, los formularios, las tablas, etc.

Con JavaScript es posible manipular y acceder a los objetos que representan el contenido de una página. De este modo, en lugar de crear solamente documentos estáticos, es posible crear documentos dinámicos. Este proceso se puede realizar gracias al método document.write().

En ejemplos anteriores, hemos visto que con el método document.write() podemos escribir un resultado por pantalla. Existen dos formas de utilizar este método para generar contenido dinámico:

* Podemos utilizarlo dentro de una secuencia de instrucciones JavaScript para mostrar un resultado en el documento de la ventana actual del navegador. Por ejemplo, es posible definir el título de una página web basándose en el nombre del sistema operativo que se utilice para abrir el documento:

```
<script type="text/javascript">
  var SO = navigator.platform;
  document.write("<h1>Documento abierto con: " + SO +
  "</h1>");
</script>
```

* La segunda forma es muy similar a la anterior. El método document.write() podemos utilizarlo para crear documentos de nuevas ventanas del navegador. Esto es una práctica muy común en las páginas web que utilizan ventanas emergentes. Los detalles de la creación y la comunicación entre ventanas las abordaremos en otro apartado. Por el momento, en el siguiente código podemos observar cómo creamos una nueva ventana sin ningún contenido, posteriormente accedemos a su respectivo

documento y finalmente, en la ventana nueva escribimos el título de la página web según el texto que ha ingresado el usuario.

```
<script type="text/javascript">
  var texto = prompt("Ingresa un titulo para la nueva
    ventana: ");
  var ventanaNueva = window.open();
  ventanaNueva.document.write("<h1>" + texto + "</h1>");
</script>
```

* La generación de código HTML a partir de código JavaScript no se limita solo a la creación de texto tal y como hemos visto en los ejemplos anteriores. Podemos crear y manipular todo tipo de objetos. El siguiente ejemplo muestra cómo generar un formulario para modificar la propiedad del color de fondo de la página:

```
<script type="text/javascript">
  document.write("<form name=\"cambiacolor\">");
  document.write("<b>Selecciona un color para el fondo de
    página:</b><br>");
  document.write("<select name=\"color\">");
  document.write("<option value=\"red\">Rojo</option>");
  document.write("<option value=\"blue\">Azul</option>");
  document.write("<option
    value=\"yellow\">Amarillo</option>");
  document.write("<option value=\"green\">Verde</option>");
  document.write("</select>");
  document.write("<input type=\"button\"
    value=\"Modifica el color\" onclick=\"document.bgColor
    =document.cambiacolor.color.value\">");
  document.write("</form>");
</script>
```

En la figura siguiente vemos el resultado de la generación de una página web dinámica. En esta página el usuario puede modificar la propiedad document.bgColor a través de un formulario HTML que se ha creado a partir de código JavaScript.

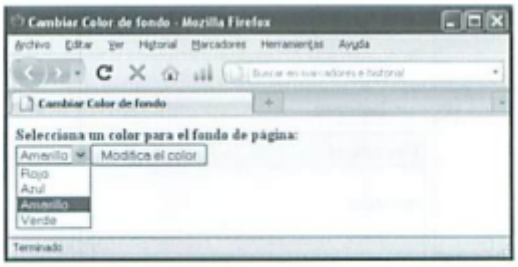

Figura 3.3. *Generación de código HTML con JavaScript. Cambio de color*