Ciclo superior:

Diseño de aplicaciones web

MÓDULO:
Desarrollo web en entorno cliente

Material original realizado por Rafael Veiga para el MEC

Adaptación y ampliación realizada por Beatriz Buyo

ÍNDICE DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE IMÁGENES

ÍNDICE DE SABER MÁS

RECOMENDACIÓN

ÍNDICE DE EJERCICIOS

ÍNDICE DE EJEMPLOS

UNIDAD DIDÁCTICA 3:

MODELO DE OBJETOS PREDEFINIDOS EN JAVASCRIPT.

1. OBJETOS DE MÁS ALTO NIVEL EN JAVASCRIPT.

Una página web, es un documento HTML que será interpretado por los navegadores de forma gráfica, pero que también va a permitir el acceso al código fuente de la misma.

El **Modelo de Objetos del Documento** (DOM) permite ver el mismo documento de otra manera, describiendo el contenido del documento como un conjunto de objetos, con los que un programa de JavaScript puede interactuar.

Según el W3C, el Modelo de Objetos del Documento es una interfaz de programación de aplicaciones (API), para documentos válidos HTML y bien construidos XML. Define la estructura lógica de los documentos y el modo en el que se acceden y se manipulan.

Ahora que ya has visto en las unidades anteriores, los fundamentos de la programación en lenguaje JavaScript, los objetos Array, las funciones y los objetos de usuario, vamos a profundizar un poco más en lo que se refiere a los objetos que podremos colocar en la mayoría de nuestros documentos.

Definimos objeto como una entidad con una serie de propiedades que definen su estado y unos métodos (funciones) que actúan sobre esas propiedades.

Como ya sabemos, la forma de acceder a una propiedad de un objeto es con el operador punto (.).

```
nombre_objeto.propiedad
```

Así mismo, la forma de acceder a un método de un objeto es:

```
nombre_objeto.metodo( [parámetros_opcionales] )
```

También podemos hacer referencia a una propiedad de un objeto, por su índice durante la creación. Los índices comienzan por 0.

Te mostramos aquí el gráfico del modelo de objetos de alto nivel, para todos los navegadores que permitan usar JavaScript

ILUSTRACIÓN 1 OBJETOS JAVASCRIPT DE ALTO NIVEL

En esta unidad nos centraremos en los objetos de alto nivel que encontrarás frecuentemente en tus aplicaciones de JavaScript: **window, screen**, **navigator**, **location**, **history** y **document.** Estudiaremos sus métodos y propiedades para que puedas comenzar a realizar tareas sencillas y para prepararte para profundizar en las propiedades y métodos, gestores de eventos, etc. que encontrarás más adelante.

1.1 OBJETO WINDOW.

En la jerarquía de objetos, tenemos en la parte superior el objeto **window**.

Este objeto está situado justamente ahí, porque es el contenedor principal de todo el contenido que se visualiza en el navegador. Tan pronto como se abre una ventana (**window**) en el navegador, incluso aunque no se cargue ningún documento en ella, este objeto **window** ya estará definido en memoria.

Además de la sección de contenido del objeto **window**, que es justamente dónde se cargarán los documentos, el campo de influencia de este objeto, abarca también las dimensiones de la ventana, así como todo lo que rodea al área de contenido: las barras de desplazamiento, barra de herramientas, barra de estado, etc.

Cómo se veía en el gráfico anterior, debajo del objeto window tenemos otros objetos como el **navigator**, **screen**, **history**, **location** y el objeto **document**. Este objeto document será el que contendrá toda la jerarquía de objetos, que tengamos dentro de nuestra página HTML.

En los navegadores más modernos, los usuarios tienen la posibilidad de abrir las páginas tanto en nuevas pestañas dentro de un navegador, como en nuevas ventanas de navegador. Para JavaScript tanto las ventanas de navegador, como las pestañas, son ambos objetos **window**.

Acceso a propiedades y métodos.

Podremos acceder a las propiedades y métodos del objeto **window** de diferentes formas, dependiendo más de nuestro estilo que de requerimientos sintácticos. Así, la forma más lógica y común de realizar esa referencia, incluye el objeto **window** tal y como se muestra en este ejemplo:

```
window.nombrePropiedad
window.nombreMétodo( [parámetros_opcionales] )
```

Como puedes ver, los parámetros van entre corchetes, indicando que son opcionales y que dependerán del método al que estemos llamando. En caso de requerirse algún parámetro los corchetes no se ponen.

Un objeto **window** también se podrá referenciar mediante la palabra **self**, cuando estamos haciendo la referencia desde el propio documento contenido en esa ventana:

```
self.nombrePropiedad
self.nombreMétodo( [parámetros_opcionales] )
```

La siguiente forma también funcionaría sin ningún problema, porque asume que las propiedades o métodos, son del objeto de mayor jerarquía (el objeto **window**) en el cuál nos encontramos.

```
nombrePropiedad
nombreMétodo( [parámetros_opcionales] )
```

```
//son ejemplos usados hasta ahora
alert("Hola");
prompt("¿Cómo estás?");

//en lugar de
window.alert("Hola");
window.prompt("¿Cómo estás?");
```

1.1.1 GESTIÓN DE VENTANAS.

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 abrir. Sin embargo, un script que se esté ejecutando en una de las ventanas principales del navegador sí podrá crear o abrir nuevas sub-ventanas.

El método que genera una nueva ventana es `window.open()`. Este método contiene tres parámetros, que definen las características de la nueva ventana. Su sintaxis es la siguiente:

```
window.open(url,nombre,características);
```

- `url`: es la URL del documento a abrir que al ser un texto tendrá que ir encerrada entre comillas,
- `nombre`: es el nombre de variable asociado a la ventana con la que podremos trabajar en el programa,
- `características`: es una lista de valores iniciales asignados a ciertas propiedades de la ventana que determinan su apariencia física (tamaño, posición, etc.). La lista irá encerrada entre comillas y los valores de las características van separados por comas.

La siguiente tabla refleja las propiedades o métodos que se pueden incluir en el parámetro `características` del método `open` del objeto `Window`.

Propiedades del argumento características del método window.open()	
Propiedad	**Descripción**
`toolbar=[yes|no]`	Muestra u oculta la barra de herramientas del navegador
`location=[yes|no]`	Muestra u oculta la barra de direcciones del navegador
`status=[yes|no]`	Muestra u oculta la barra de estado del navegador
`menubar=[yes|no]`	Muestra u oculta la barra de menús
`scrollbars=[yes|no]`	Muestra u oculta las barras de desplazamiento
`resizable=[yes|no]`	Permite o no al usuario redimensionar el tamaño de la ventana
`width=nnn`	Define número de píxeles del ancho de la ventana
`height=nnn`	Define número de píxeles del alto de la ventana
`left=nnn`	Nº de píxeles que hay desde el lado izquierdo de la pantalla a la ventana
`top=nnn`	Nº de píxeles que hay desde el lado superior de la pantalla a la ventana

TABLA 1 PROPIEDADES DEL ARGUMENTO CARACTERÍSTICAS DEL MÉTODO WINDOW.OPEN()

La siguiente instrucción abre una nueva ventana de `800` píxeles de ancho y `600` píxeles de alto que contiene un documento HTML ("`nueva.html`") que se encuentra ubicado en la misma ruta. El nombre por el que se podrá identificar a la ventana en el código es "`nueva`" y para acceder a los métodos y propiedades de esta nueva ventana lo haremos por la referencia asignada al objeto `subVentana`.

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

Si se desea omitir alguno de los parámetros bastará con dejar únicamente las dos comillas dobles. También se puede utilizar el `1` y el `0` en lugar de las palabras `yes` y `no` (respectivamente).

Lo importante de la instrucción anterior es la declaración e inicialización de la variable `subVentana` con el resultado de la llamada al método `open()`. De esta forma podremos hacer referencias a esta nueva ventana en nuestro código desde el script de la ventana principal. Por ejemplo, si quisiéramos cerrar la nueva ventana desde nuestro script, simplemente tendríamos que hacer: `subVentana.close();` lo que resulta necesario ya que si escribiéramos `window.close()`, `self.close()` o `close()` estaríamos intentando cerrar nuestra propia ventana (previa confirmación), pero no la **`subVentana`** que creamos con la instrucción del recuadro.

RECOMENDACIÓN:

En el código del ejemplo de la página anterior se define una función que crea una ventana almacenada en la variable nuevaVentana.

El siguiente código escribe directamente el código html en el documento de la nueva ventana accediendo directamente al objeto document del objeto window a través del método write del objeto document.

```
function  crearVentana() {
   var nuevaVentana  =
window.open("","","toolbar=yes,location=yes,directories=yes,status=yes,menubar=yes,scrol
lbars=yes,resizable=yes,height=100,width=800,left=100,top=100");

   //comprobamos que existe la ventana
   //ya que pueden estar bloqueados los elementos emergentes
   if (nuevaVentana) {
        // cabecera del documento html de la ventana creada
        nuevaVentana.document.write("<!DOCTYPE html>\n");
        nuevaVentana.document.write("<html>\n");
        nuevaVentana.document.write("   <head>\n");
        nuevaVentana.document.write("     <meta http-equiv=\"Content-Type\"
content=\"text/html;charset=utf-8\">\n");
        nuevaVentana.document.write("     <title>Ejercicio 1. Nueva ventana</title>\n");
        nuevaVentana.document.write("   </head>\n");

        // cuerpo del documento
        nuevaVentana.document.write("   <body>\n");
        nuevaVentana.document.write("     <div>\n");

        // propiedades configurables de la ventana creada con el método open
        nuevaVentana.document.write("         toolbar=yes<br />\n");
        nuevaVentana.document.write("         location=yes<br />\n");
        nuevaVentana.document.write("         directories=yes<br />\n");
        nuevaVentana.document.write("         status=yes<br />\n");
        nuevaVentana.document.write("         menubar=yes<br />\n");
        nuevaVentana.document.write("         scrollbars=yes<br />\n");
        nuevaVentana.document.write("         resizable=yes<br />\n");
        nuevaVentana.document.write("         height=400<br />\n");
        nuevaVentana.document.write("         width=800<br />\n");
        nuevaVentana.document.write("         left=100<br />\n");
        nuevaVentana.document.write("         top=100<br />\n");
        nuevaVentana.document.write("      </div>\n");
        nuevaVentana.document.write("   </body>\n");
        nuevaVentana.document.write("</html>");
   } else {
        alert("Revisa si tienes bloqueados los pop-ups o ventanas emergentes");
   }
}
```

Escribe el código JavaScript anterior en un archivo Ejercicio1.js.

Escribe ahora el siguiente código en un archivo Ejercicio1.html que enlace con el archivo js anterior.

```
<div><a href="" onmouseover="crearVentana()">Pasa por encima de este texto</a></div>
```

Esta línea puesta en un documento html muestra un enlace al cual se le ha asignado un evento de pasar por encima el ratón. *(No es una buena solución pero es lo que podemos hacer hasta ahora)*

Abre el archivo html en el navegador.

Cuando pasas el ratón por encima del enlace se ejecuta la función crearVentana que crea la ventana con el contenido indicado por la función.

Pruébalo en distintos navegadores observa el código fuente que ha generado (te será de gran ayuda cuando estudies el modelo DOM) y trata de contestar a las siguientes preguntas:

1. ¿Se ve la barra de estado?
2. ¿En qué navegador se genera un código fuente más correcto y fiel al código escrito?
3. ¿Sale en todos los navegadores la ventana de alerta cuando están bloqueados los pop-ups?
4. ¿Qué diferencia hay en los distintos navegadores si asignamos `no` en lugar de `yes` a las propiedades? ¿y si se omiten todas las propiedades?

RECOMENDACIÓN 1 CONFIGURACIÓN DE LA VENTANA CREADA

RECOMENDACIÓN:

Escribe el siguiente código JavaScript, pruébalo en distintos navegadores y contesta a la pregunta: ¿cuál es la razón por la que no se cierra la primera ventana?

```
function  crearNueva(direccion) {//direccion contiene la URL que se abrirá en la ventana creada
        return window.open(direccion,"","height=400,width=800");
}
function  cerrarNueva() {
        if  (nuevaVentana) {//nuevaVentana es una variable global
                nuevaVentana.close();
        }
}
var nuevaVentana;
var direccion="http://www.google.es";
alert("Vamos a crear una nueva ventana con la dirección de google");
crearNueva(direccion);
alert("¿Has visto ya la ventana? \nAhora vamos a abrir otra con la dirección del tiempo,
así que mueve antes la anterior a otra zona de la pantalla");
direccion="http://www.eltiempo.es";
crearNueva(direccion);
alert("Ahora tenemos tres ventanas abiertas: \n1. La que está ejecutando el script\n2.
La de la página de google\n3. La que nos muestra el tiempo\n\nAhora vamos a llamar al
método de cerrar la ventana"); //alert("Todo en la misma línea")
cerrarNueva();
alert("¿Cuál de ellas se cerró?\n\nLlamamos de nuevo al método cerrarNueva");
cerrarNueva();
```

¿Cuál es el problema de este ejercicio?
¿Cómo lo solucionarías?

RECOMENDACIÓN 2 GESTIÓN DE VENTANAS

En el enlace siguiente puedes consultar la información referente al método window.open().

Hay otras propiedades que no están reflejadas en la lista de la tabla 1 por no estar implementadas en todos los navegadores (centerscreen, outerHeight,…).

Método window.open()

1.1.2 PROPIEDADES DEL OBJETO WINDOW.

El objeto `window` representa una ventana abierta en un navegador. Si un documento contiene marcos (`<frame>` o `<iframe>`), el navegador crea un objeto `window` para el documento HTML, y un objeto `window` adicional para cada marco. El elemento `frame` está obsoleto en HTML5.

Propiedades del objeto Window	
Propiedad	**Descripción**
`closed`	Devuelve un valor `Boolean` indicando cuando una ventana ha sido cerrada o no.
`console`	Retorna una referencia a un objeto Console, que tiene métodos para mostrar información al usuario a través de la consola del ordenador. (Ej: console.log("Hola"))
`document`	Devuelve el objeto `document` de la ventana.
`frameElement`	Devuelve el elemento `iframe` en el que está insertado el documento, null si es la ventana principal.
`frames`	Devuelve un array con todos `iframes` de la ventana actual.
`history`	Devuelve el objeto `history` de la ventana.
`innerHeight`	Altura útil de la ventana. No incluye barras de herramientas, estado, desplazamiento.
`innerWidth`	Anchura útil de la ventana
`length`	Devuelve el nº de `iframes` que hay en una ventana. (También `frames.length`)
`localStorage`	Permite almacenar datos localmente de forma permanente. ***(Se verá en otra unidad)***
`location`	Devuelve la URL de la barra de direcciones de la ventana.
`name`	Ajusta o devuelve el nombre de una ventana.
`navigator`	Devuelve el objeto `navigator` de una ventana.
`opener`	Devuelve el objeto `window` que abrió la ventana actual.
`outerHeight`	Altura total de la ventana.
`outerWidth`	Anchura total de la ventana.
`pageXOffset`	Devuelve los píxeles que está desplazado horizontalmente el documento actual desde la esquina superior izquierda.
`pageYOffset`	Devuelve los píxeles que está desplazado verticalmente el documento actual desde la esquina superior izquierda.
`parent`	Devuelve el objeto `window` padre de la ventana actual.
`screen`	Devuelve el objeto `screen` de la ventana.
`screenLeft`	Muestra el nº de píxeles desde el borde izquierdo de la pantalla al borde izquierdo de la ventana. (Excepto Firefox)
`screenTop`	Muestra el nº de píxeles desde el borde superior de la pantalla al borde superior de la ventana. (Excepto Firefox)
`screenX`	Muestra el nº de píxeles desde el borde izquierdo de la pantalla al borde izquierdo de la ventana. (Firefox)
`screenY`	Muestra el nº de píxeles desde el borde superior de la pantalla al borde superior de la ventana. (Firefox)
`sessionStorage`	Almacena datos localmente durante la sesión. ***(Se verá en otra unidad)***
`scrollX`	Alias de `pageXOffset`.
`scrollY`	Alias de `pageYOffset`.
`self`	Devuelve la ventana actual.
`status`	Obtiene o establece el texto mostrado en la barra de estado.
`top`	Devuelve el objeto `window` de nivel superior.

TABLA 2 PROPIEDADES DEL OBJETO WINDOW

1.1.3 MÉTODOS DEL OBJETO WINDOW.

Ya hemos usado alguno de los métodos (`open`, `alert`, `close`, `prompt`) del objeto `window` en apartados anteriores. Son muchos los métodos de que dispone este objeto, aunque no todos funcionan en todos los navegadores.

Métodos del objeto Window	
Método	**Descripción**
`alert(mens)`	Muestra una ventana emergente de alerta con el mensaje y un botón de aceptar.
`atob(cadena)`	Decodifica una cadena codificada con `btoa(cadena)`;
`blur()`	Elimina el foco de la ventana actual.
`btoa(cadena)`	Codifica una cadena en base-64.
`clearInterval(id)`	Resetea el cronómetro ajustado con `setInterval()` identificado por `id`.
`clearTimeout(id)`	Resetea el cronómetro ajustado con `setTimeout()` identificado por `id`.
`close()`	Cierra la ventana actual.
`confirm(mens)`	Muestra una ventana emergente con un mensaje, un botón de aceptar y un botón de cancelar.
`focus()`	Coloca el foco en la ventana actual.
`getCompusedStyle()`	Obtiene las propiedades y valores CSS actuales de un elemento concreto.
`isNaN(n)`	Verdad si `n` es un NaN (se verá en el objeto Number).
`matchMedia(mQS)`	Obtiene u objeto MediaQueryList que representa el resultado del Media Query Search especificado como argumento.
`moveBy(±n,±m)`	Mueve la ventana que tiene el foco `n` píxeles horizontalmente y `m` píxeles verticalmente.
`moveTo(x,y)`	Mueve la ventana que tiene el foco a la coordenada `(x,y)`.
`open(url,nom,car)`	Abre una nueva ventana de navegación.
`print()`	Imprime la página.
`prompt(mens,val)`	Muestra una ventana de diálogo para introducir datos con un mensaje `mens` y un valor `val` por defecto.
`resizeBy(±n, ±m)`	Modifica el tamaño de la ventana en `n` píxeles de ancho y `m` píxeles de alto.
`resizeTo(n,m)`	Establece el tamaño de la ventana en `n` píxeles de ancho y `m` píxeles de alto.
`scrollBy(±x,±y)`	Mueve el contenido en la ventana que tiene el foco `n` píxeles horizontalmente y `m` píxeles verticalmente.
`scrollTo(x,y)`	Desplaza el contenido de la ventana a las coordenadas `(x,y)` de la misma.
`setInterval(ex,t)`	Evalúa una expresión `ex` después de `t` milisegundos. Devuelve un identificador.
`setTimeOut(ex,t)`	Evalúa una expresión `ex` después de `t` milisegundos. Devuelve un identificador.
`stop()`	Realiza la función de parada de la ventana del navegador.

TABLA 3 MÉTODOS DEL OBJETO WINDOW

Otros métodos también interesantes son `showModalDialog()` y `showModelessDialog()` que permiten abrir nuevas ventanas configurables pero que no funcionan en todos los navegadores (de hecho, están pensadas para Internet Explorer por lo que el modo Modal no se comporta de la misma forma en otros navegadores). Chrome a ha decidido eliminar este método definitivamente a partir de la versión 43 y Firefox anunció la eliminación del soporte a partir de la versión 46.

PARA SABER MÁS

En el siguiente enlace puedes consultar la información sobre los métodos mencionados en el párrafo anterior y cómo resuelve el problema para los distintos navegadores.

Apertura de ventanas

PARA SABER MÁS 1 MÉTODOS DEL OBJETO WINDOW

RECOMENDACIÓN

En el siguiente enlace tienes más información de las propiedades, métodos y colecciones de este objeto. Desde este enlace podrás comprobar su funcionamiento con sencillo ejemplos.

Propiedades, métodos y colecciones del objeto Window en w3schools.

Propiedades y métodos del objeto Window en MDN web docs moz://a

RECOMENDACIÓN 3 PROPIEDADES, MÉTODOS Y COLECCIONES DEL OBJETO WINDOW

1.2 OBJETO DOCUMENT

Cada documento cargado en una ventana del navegador, será un objeto de tipo `document`.

El objeto `document` proporciona a los scripts, el acceso a todos los elementos HTML dentro de una página.

Este objeto forma parte además del objeto `window` y puede ser accedido a través de la propiedad `window.document` o directamente `document` (ya que podemos omitir la referencia al objeto `window` actual).

El diagrama siguiente muestra la jerarquía de los objetos de alto nivel del JavaScript.

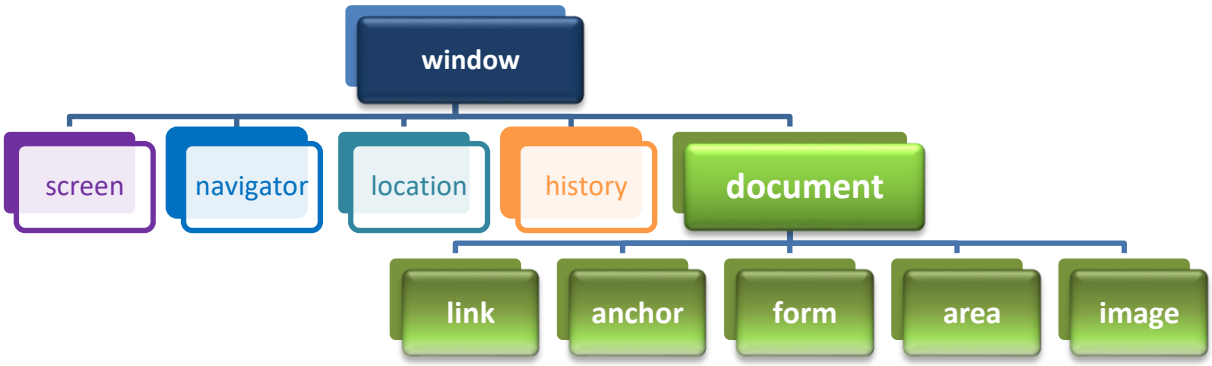

ILUSTRACIÓN 2 JERARQUÍA DE OBJETOS DE JAVASCRIPT

Propiedades del objeto Document	
Propiedad	**Descripción**
`activeElement`	Devuelve el elemento del DOM que tiene el foco.
`baseURI`	Devuelve el valor de la URI. Sólo IE.
`body`	Devuelve el `body` de un documento.
`characterSet`	Devuelve el conjunto de caracteres en el que está codificado el documento.
`cookie`	Devuelve todos los nombres/valores de las cookies asociadas al documento.
`doctype`	Devuelve la declaración `DOCTYPE` del documento.
`documentElement`	Devuelve el elemento `<html>` del documento.
`documentMode`	Devuelve el valor usado por el navegador para renderizar el documento. Sólo IE. (5, 7, 8, 9, 10, 11). Si no se especifica `DOCTYPE` IE devuelve 5.
`documentURI`	Establece o devuelve el valor de la URI. Todos excepto IE.
`domain`	Devuelve el nombre del dominio del servidor que llamó al documento.
`head`	Devuelve el elemento `head` del documento.
`inputEncoding`	Lo mismo que `characterSet`.
`lastModified`	Devuelve la fecha y la hora en la que el documento fue modificado.
`readyState`	Devuelve el estado de carga de un documento: • `uninitialized` – No comenzó la descarga • `loading` – Se está descargando • `loaded` – Ha sido descargado • `interactive` – Descargado lo suficiente para que el usuario interactúe • `complete` – Totalmente descargado
`referrer`	Devuelve la URL del documento que descargó el documento actual.
`title`	Devuelve o establece el título del documento.
`URL`	Devuelve la URL completa del documento.

TABLA 4 PROPIEDADES DEL OBJETO DOCUMENT

Métodos del objeto Document	
Método	**Descripción**
`addEventListener()`	Añade un gestor de eventos al documento.
`adoptNode()`	Adopta un nodo de otro documento. (Y todos sus descendientes) El nodo es borrado del documento original.
`close()`	Cierra el flujo abierto previamente con `document.open()`.
`createAttribute()`	Crea un atributo de un nodo.
`createComment()`	Crea un nodo comentario con un texto.
`createDocumentFragment()`	Crea un nodo fragmento de documento vacío. Sirve para extraer, modificar trozos del documento original y/o añadir posteriormente al mismo.
`createElement()`	Crea un nodo elemento.
`createTextNode()`	Crea un nodo de texto.
`getElementById(id)`	Devuelve el elemento con dicho `id`.
`getElementsByClassName(className)`	Devuelve los elementos con dicho `className`.
`getElementsByName(name)`	Devuelve los elementos con dicho `name`.
`getElementsByTagName(tagName)`	Devuelve los elementos con dicho `tagName`.
`hasFocus()`	Indica si el documento tiene el foco.
`importNode(nodo,descendientes)`	Importa un nodo de otro documento sin borrarlo del original. Si el parámetro descendientes es `true` importa también sus descendientes.
`normalize()`	Borra los nodos texto vacíos y une los adyacentes.
`open()`	Abre un flujo de salida para ordenes write y/o writeln que se visualizan en el documento cuando se cierra con close().
`querySelector()`	Devuelve el primer elemento cuyo selector concuerda con el especificado.
`querySelectorAll()`	Lo mismo que la anterior preo crea un nodo lista con todos los que concuerdan.
`removeEventListener()`	Borra un gestor de eventos añadido previamente con `addEventListener()`.
`write(expresión)`	Escribe expresiones HTML y/o código JavaScript en el documento.
`writeln(expresion)`	Como `write()` pero añade al final un salto de línea.

TABLA 5 MÉTODOS DEL OBJETO DOCUMENT

Colecciones del objeto Document	
Colección	**Descripción**
`anchors`	Contiene todas las anclas del documento (zonas etiquetadas con `<a name="…">`).
`embeds`	Contiene todos los elementos embebidos del documento (etiqueta `<embed>`).
`forms`	Contiene todos los formularios del documento (etiqueta `<form …>`).
`images`	Contiene todas las imágenes del documento (etiqueta `<img …>`).
`links`	Contiene todos los enlaces del documento (etiqueta `<a href="…">`).
`scripts`	Contiene todos los script del documento (etiqueta `<script>`).

TABLA 6 COLECCIONES DEL OBJETO DOCUMENT

RECOMENDACIÓN

En el siguiente enlace tienes más información de las propiedades, métodos y colecciones de este objeto. Desde este enlace podrás comprobar su funcionamiento con sencillo ejemplos.

Propiedades, métodos y colecciones del objeto Document.

RECOMENDACIÓN 4 PROPIEDADES, MÉTODOS Y COLECCIONES DEL OBJETO DOCUMENT

RECOMENDACIÓN:

Sirviéndote de los ejemplos del enlace recomendado inventa un ejemplo que ponga a prueba al menos 4 propiedades y 5 métodos y que use 3 de las colecciones.

En el código comenta numerando las propiedades, los métodos y las colecciones que pones a prueba.

Esta vez y mientras no hayamos estudiado la gestión de eventos, enlaza el script justo antes de finalizar el body. De esta forma garantizamos que los objetos son conocidos en el momento en que el script empieza a ejecutarse.
Si se enlazase en la cabecera del html el script se ejecutaría sin conocer todavía todos los objetos.

RECOMENDACIÓN 5 PROPIEDADES, MÉTODOS Y COLECCIONES DEL OBJETO DOCUMENT

1.3 OBJETO SCREEN

Este objeto permite obtener información sobre las características de la ventana del navegador: la altura y la anchura de la pantalla en píxeles (total y útil), la profundidad del color de la paleta de colores del navegador y la profundidad de color de cada píxel de la pantalla.

Propiedades del objeto Screen	
Propiedad	**Descripción**
`availHeight`	Altura útil en píxeles de la pantalla.
`availWitdh`	Anchura útil en píxeles de la pantalla.
`colorDepth`	Número de bits para representar cada color de la paleta de colores del navegador.
`height`	Altura total en píxeles de la pantalla.
`pixelDepth`	Número de bits para representar el color de cada píxel de la ventana.
`width`	Anchura total en píxeles de la pantalla.

TABLA 7 PROPIEDADES DEL OBJETO SCREEN

La altura útil es menor que la total debido al espacio ocupado por las barras de títulos y/o herramientas que cada navegador tiene configuradas.

Estas propiedades resultan de utilidad a las personas que se dedican al diseño web para adaptar sus diseños a las características de la pantalla del usuario.

Este objeto no tiene ningún método.

RECOMENDACIÓN:

Escribe el siguiente código JavaScript, pruébalo al menos en tres navegadores distintos (Explorer, Mozilla, Chrome, Opera, Safari, …) y contesta después a las preguntas:

```
document.write("Propiedades del objeto -screen-<br />");
document.write("screen.height = "+screen.height+"<br />");
document.write("screen.width = "+screen.width+"<br />");
document.write("screen.colorDepth = "+screen.colorDepth+"<br />");
document.write("screen.pixelDepth = "+screen.pixelDepth+"<br />");
document.write("screen.availHeight = "+screen.availHeight+"<br />");
document.write("screen.availWidth = "+screen.availWidth+"<br />");
```

1. ¿Es la profundidad de color la misma en todos los navegadores?
2. ¿Cambian los valores de las propiedades en función del zoom establecido en el navegador?

   Primero haz zoom y luego refresca. Hazlo varias veces por encima y por debajo del 100%.

Comenta las incidencias en el foro de la unidad.

RECOMENDACIÓN 6 PROPIEDADES DEL OBJETO SCREEN

RECOMENDACIÓN

En el siguiente enlace tienes más información de las propiedades, métodos y colecciones de este objeto. Desde este enlace podrás comprobar su funcionamiento con sencillo ejemplos.

Propiedades, métodos y colecciones del objeto Screen.

RECOMENDACIÓN 7 PROPIEDADES, MÉTODOS Y COLECCIONES DEL OBJETO SCREEN

1.4 OBJETO NAVIGATOR

El objeto `navigator` contiene información sobre el navegador que estamos utilizando cuando abrimos una URL o un documento local.

Propiedades del objeto Navigator	
Propiedad	**Descripción**
`appCodeName`	Cadena que contiene el nombre en código del navegador.
`appName`	Cadena que contiene el nombre del cliente.
`appVersion`	Cadena que contiene información sobre la versión del cliente.
`cookieEnabled`	Determina si las cookies están o no habilitadas en el navegador.
`geolocation`	Devuelve un objeto Geolocation que puede ser usado para localizar la posición del usuario. Pide permiso.
`language`	Devuelve el lenguaje del navegador.
`onLine`	Determina cuando el navegador está en línea.
`platform`	Cadena con la plataforma sobre la que se está ejecutando el programa cliente.
`plugins`	Colección de elementos añadidos en el navegador.
`userAgent`	Cadena que contiene la cabecera completa del agente, enviada en una petición HTTP. Contiene la información de las propiedades `appCodeName` y `appVersion`.

TABLA 8 PROPIEDADES DEL OBJETO NAVIGATOR

Métodos del objeto Navigator	
Método	**Descripción**
`javaEnabled()`	Devuelve true si el cliente permite la utilización de Java, en caso contrario, devuelve false.

TABLA 9 MÉTODOS DEL OBJETO NAVIGATOR

RECOMENDACIÓN

En el siguiente enlace tienes más información de las propiedades, métodos y colecciones de este objeto. Desde este enlace podrás comprobar su funcionamiento con sencillo ejemplos.

Propiedades, métodos y colecciones del objeto Navigator.

RECOMENDACIÓN 8 PROPIEDADES, MÉTODOS Y COLECCIONES DEL OBJETO NAVIGATOR

RECOMENDACIÓN:

El siguiente código JavaScript hace un listado de todas las propiedades del objeto Navigator y los valores que tienen en función del navegador donde se ejecute. Pruébalo en diferentes navegadores para ver las diferentes propiedades que muestran cada uno de ellos. Son bastantes más de las indicadas en la tabla.

```
document.write("<h1>Valores de las propiedades del objeto Navigator</h1>\n<ul>\n");
for ( var prop in navigator ) {
   document.write("<li>"+prop+"-------->"+navigator[prop]+"</li>\n");
   if ( prop=="plugins" ) {
      document.write("<ul>\n");
      for ( var i=0; i<navigator.plugins.length; i++ ) {
         document.write("<li>Plug-in numero "+i+"</li>\n<ul>\n");
         for ( var prop in navigator.plugins[i] ) {
            document.write("<li>"+prop+"--->"+navigator.plugins[i][prop]+"</li>\n");
         }
         document.write("</ul>\n");
      }
      document.write("</ul>\n");
   } // if prop==plugins
}
document.write("</ul>\n");
```

RECOMENDACIÓN 9 PROPIEDADES DEL OBJETO NAVIGATOR

1.5 OBJETO LOCATION

El objeto `location` contiene información referente a la URL actual.

Este objeto, es parte del objeto `window` y accedemos a él a través de la propiedad `window.location`.

Propiedades del objeto Location	
Propiedad	**Descripción (* - E/S)**
`hash`	(*)Cadena que contiene el nombre del enlace (#....), dentro de la URL.
`host`	(*)Cadena que contiene el nombre del servidor y el número del puerto, en la URL.
`hostname`	(*)Cadena que contiene el nombre de dominio del host (o la dirección IP), en la URL.
`href`	(*)Cadena que contiene la URL completa.
`origin`	Devuelve el protocolo, nombre del host y puerto de una URL.
`pathname`	(*)Cadena que contiene el camino al recurso en una URL.
`port`	(*)Cadena que contiene el número de puerto del servidor en una URL.
`protocol`	(*)Cadena que contiene el protocolo utilizado (incluyendo los dos puntos), dentro de la URL.
`search`	(*)Cadena de búsqueda dentro de la URL.

TABLA 10 PROPIEDADES DEL OBJETO LOCATION

Métodos del objeto Location	
Método	**Descripción**
`assign()`	Carga un nuevo documento.
`reload()`	Vuelve a cargar la URL especificada en la propiedad `href` del objeto `location`.
`replace(url)`	Reemplaza el historial actual mientras carga la URL especificada en URL.

TABLA 11 MÉTODOS DEL OBJETO LOCATION

RECOMENDACIÓN

En el siguiente enlace tienes más información de las propiedades, métodos y colecciones de este objeto. Desde este enlace podrás comprobar su funcionamiento con sencillo ejemplos.

Propiedades, métodos y colecciones del objeto Location.

RECOMENDACIÓN 10 PROPIEDADES, MÉTODOS Y COLECCIONES DEL OBJETO LOCATION

1.6 OBJETO HISTORY

El objeto `history` contiene información referente a las URLs visitadas por el usuario con un navegador.

Este objeto, es parte del objeto `window` y accedemos a él a través de la propiedad `window.history`.

Propiedades del objeto History	
Propiedad	**Descripción**
`length`	Número de URLs visitadas en el historial del navegador.

TABLA 12 PROPIEDADES DEL OBJETO HISTORY

Métodos del objeto History	
Método	**Descripción**
`back()`	Carga la URL previa en el historial.
`forward()`	Carga la URL siguiente en el historial.
`go()`	Carga una URL específica del historial.

TABLA 13 MÉTODOS DEL OBJETO HISTORY

RECOMENDACIÓN

En el siguiente enlace tienes más información de las propiedades, métodos y colecciones de este objeto. Desde este enlace podrás comprobar su funcionamiento con sencillo ejemplos.

Propiedades, métodos y colecciones del objeto History.

RECOMENDACIÓN 11 PROPIEDADES, MÉTODOS Y COLECCIONES DEL OBJETO HISTORY

RECOMENDACIÓN:

Puedes buscar ejemplos y ayuda en Internet que te faciliten la tarea.
Completa el ejercicio comenzado en el ejercicio 3 con:

- 4 propiedades y 2 métodos del objeto location.
- 1 propiedad y 2 métodos del objeto history.

RECOMENDACIÓN 12 PROPIEDADES Y MÉTODOS DEL OBJETO LOCATION E HISTORY

2. MARCOS.

Antes de la llegada de las hojas de estilo en cascada y la utilización de las capas para la maquetación de los elementos de una página web las personas dedicadas al diseño de interfaces web empleaban las tablas y/o los marcos.

La ventaja que suponía el uso de los marcos sobre las tablas es que permitía independizar los diferentes contenidos mostrados en la misma página ya que el contenido de cada marco se almacenaba en un documento html.

El uso de los marcos implicaba tener una página principal de definición de marcos en la cual se indicaba la distribución, tamaño y documento que enlazaría cada uno de ellos. El siguiente ejemplo muestra un documento html de definición de dos marcos de columna: la primera ocupará un 20% del ancho de la ventana del navegador y la segunda el 80% restante (que se podría haber puesto como un *).

```
<!DOCTYPE HTML>
<html>
   <head>
      <title>Documento de definición de marcos</title>
   </head>
   <frameset cols="20%, 80%">
      <frame src="indiceContenidos.html" title="MarcoIzquierdo" id="MI" name="MarcoIzq" />
      <frame src="contenidos.html" title="MarcoDerecho" id="MD" name="MarcoDer" />>
   </frameset>
</html>
```

EJEMPLO 1 DOCUMENTO DE DEFINICIÓN DE MARCOS

Un objeto `frame`, representa un marco HTML. La etiqueta `<frame>` identifica una ventana particular, dentro de un conjunto de marcos (`frameset`). Para cada etiqueta `<frame>` se creará un objeto `frame`.

Además, cualquier documento html normal (con `body`) puede incluir uno o más marcos inline, lo que se conoce como `iframe` para cada uno de los cuales se crearía también un objeto `frame`.

La etiqueta frame no está soportada en HTML5.

2.1 PROPIEDADES Y MÉTODOS DEL OBJETO FRAME/IFRAME.

Propiedades del objeto Frame/Iframe	
Propiedad	**Descripción (* - No HTML5 para etiqueta frame ** - Tampoco para iframe)**
`align`	(**) Cadena que contiene el valor del atributo `align` (alineación) en un `iframe`.
`contentDocument`	Devuelve el objeto documento contenido en un `frame/iframe`.
`contentWindow`	Devuelve el objeto `window` generado por un `frame/iframe`.
`frameBorder`	(**) Cadena que contiene el valor del atributo `frameborder` (borde del marco) de un `frame/iframe`.
`height`	Cadena que contiene el valor del atributo `height` (altura) de un `iframe`.
`longDesc`	(**) Cadena que contiene el valor del atributo `longdesc` (descripción larga) de un `frame/iframe`.
`marginHeight`	(**) Cadena que contiene el valor del atributo `marginheight` (alto del margen) de un `frame/iframe`.
`marginWidth`	(**) Cadena que contiene el valor del atributo `marginwidth` (ancho del margen) de un `frame/iframe`.
`name`	(*) Cadena que contiene el valor del atributo `name` (nombre) de un `frame/iframe`.
`noResize`	(*) Cadena que contiene el valor del atributo `noresize` de un `frame/iframe`.
`sandbox`	Habilita un conjunto de restricciones extras para el contenido de un `iframe`. allow-forms allow-pointer-lock allow-popups allow-same-origin allow-scripts allow-top-navigation
`scrolling`	(**) Cadena que contiene el valor del atributo `scrolling` (desplazamiento) de un `frame/iframe`.
`src`	(*) Cadena que contiene el valor del atributo `src` (origen) de un `frame/iframe`.
`width`	Cadena que contiene el valor del atributo `width` (ancho) de un `iframe`.

TABLA 14 PROPIEDADES DEL OBJETO FRAME/IFRAME

Métodos del objeto Frame/Iframe	
Método	**Descripción**
`onload()`	Script que se ejecutará inmediatamente después de que se cargue el `frame/iframe`.

TABLA 15 MÉTODOS DEL OBJETO FRAME/IFRAME

RECOMENDACIÓN

En el siguiente enlace puedes comprobar el funcionamiento de las propiedades de los frame/iframe.

En el siguiente enlace tienes más información sobre los atributos de la propiedad sanbox.

Atributos.

RECOMENDACIÓN 13 FRAMES (W3C)

Recuerda que los objetos `frame` forman parte de la propiedad `frames` del objeto `window`. Es una colección con todos los marcos e `iframes` que tenga un documento.

Por razones de seguridad, los contenidos de un documento solo pueden ser accedidos desde otro documento si ambos están en el mismo dominio.

2.2 JERARQUÍAS.

Uno de los aspectos más atractivos de JavaScript en las aplicaciones cliente, es que permite las interacciones del usuario con un marco o ventana, que provocarán actuaciones en otros marcos o ventanas. En esta sección te daremos algunas nociones para trabajar con múltiples ventanas y marcos.

Marcos: Padres e Hijos.

En el gráfico de jerarquías de objetos, viste como el objeto `window` está en la cabeza de la jerarquía y puede tener sinónimos como `self`. En esta sección veremos que, cuando trabajamos con marcos o iframes, podemos referenciar a las ventanas como: `frame`, `top` y `parent`.

Aunque el uso de marcos o iframes es completamente válido en HTML, en términos de usabilidad y accesibilidad no se recomiendan, por lo que su uso está en verdadero declive. El problema fundamental con los marcos, es que las páginas contenidas en esos marcos no son directamente accesibles, en el sentido de que, si navegamos dentro de los frames, la URL principal de nuestro navegador no cambia, con lo que no tenemos una referencia directa de la página en la que nos encontramos. Esto incluso es mucho peor si estamos accediendo con dispositivos móviles. Otro problema con los frames es que los buscadores como Google, Bing, etc., no indexan bien los frames, en el sentido de que si por ejemplo registran el contenido de un frame, cuando busquemos ese contenido, nos conectará directamente con ese frame como si fuera la página principal, con lo que la mayoría de las veces perdemos la referencia de la sección del portal o web en la que se encuentra el marco.

Si vuelves a mirar el Ejemplo 1 verás que el `frameset` establece las relaciones entre los marcos de la colección. El frameset se cargará en la ventana principal (ventana padre), y cada uno de los marcos (frames), definidos dentro del frameset, será un marco hijo (ventanas hijas).

La figura de la derecha trata de ilustrar la jerarquía que hay entre dichos objetos. En ella puedes observar que la ventana padre, la que contiene el frameset, no tiene ningún objeto `document` (ya que el frameset no puede contener los objetos típicos del HTML como formularios, controles, etc.), mientras que los frames hijos sí tienen un objeto `document`.

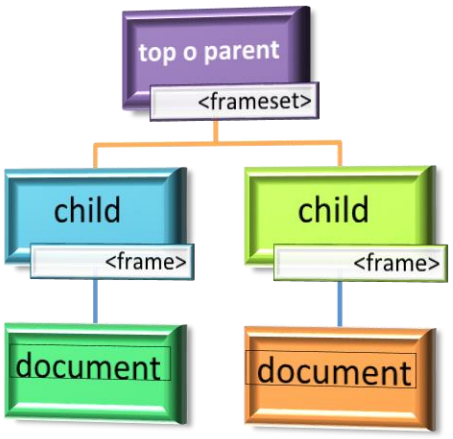

ILUSTRACIÓN 3 JERARQUÍA DE FRAMES

El objeto `document` de un marco, es independiente del objeto `document` del otro marco. En realidad, cada uno de los marcos, será un objeto `window` independiente.

2.3 COMUNICACIÓN ENTRE MARCOS.

La ventaja de la maquetación de un sitio web usando marcos es que con el lenguaje JavaScript podemos establecer una comunicación entre los diferentes marcos. Para hacerlo es indispensable que desde cualquier marco podamos referenciar a los demás, pudiendo establecer una comunicación entre padre a hijos y entre hermanos.

Comunicación entre Padre e Hijos.

Desde el momento en el que el documento padre contiene uno o más marcos, ese documento padre mantiene un array con sus marcos hijo a los que puede referenciar:

- usando las propiedades y métodos del array,
- por el nombre que le hemos dado a ese marco,
- por el id,
- por el atributo name que hemos puesto en la etiqueta <frame>.

Ejemplos de referencias a los marcos hijo: *[window.] es opcional por eso está escrito entre [])*

EJEMPLO 2 REFERENCIAS PADRE-HIJO

```
[window.]frames[n].objeto-función-variable-nombre
[window.]frames["nombreDelMarco"].objeto-función-variable-nombre
[window.]nombreDelMarco.objeto-función-variable-nombre
```

El índice numérico n que indica el número de frame, está basado en el orden en el que aparecen en el documento frameset. Se recomienda que pongamos un nombre a cada frame en dicho documento ya que así la referencia a utilizar será mucho más intuitiva.

Comunicación entre Hijos y Padre.

Es bastante más común enlazar scripts al documento padre (frameset), ya que éste se carga una vez y permanecerá cargado con los mismos datos, aunque hagamos modificaciones dentro de los marcos.

Desde el punto de vista de un documento hijo (aquel que está en un frame), su antecesor en la jerarquía será denominado el padre (`parent`). Por lo tanto, para hacer referencia a elementos del padre se hará utilizando la siguiente sintaxis:

```
parent.objeto-función-variable-nombre
```

EJEMPLO 3 REFERENCIAS HIJO-PADRE

Si el elemento al que accedemos en el padre es una función que devuelve un valor, el valor devuelto será enviado al hijo sin ningún tipo de problemas. Por ejemplo:

```
var valor=parent.nombreFuncion();
//o también
var valor=top.nombreFuncion();
```

EJEMPLO 4 OBTENER UN VALOR DEVUELTO POR UNA FUNCIÓN DEL PADRE

Comunicación entre Hermanos.

El navegador necesita un poco más de asistencia cuando queremos que una ventana hija se comunique con una de sus hermanas. Una de las propiedades de cualquier ventana o marco es su padre (parent – el cuál será null cuando estamos hablando de una ventana sin hijos). Por lo tanto, la forma de comunicar dos ventanas o marcos hermanos va a ser siempre referenciándolos a través de su padre, ya que es el único nexo de unión entre ambos (los dos tienen el mismo padre). Para ello empleamos los mismos formatos del Ejemplo 2 pero sustituyendo `[window.]` por `parent.`

2.4 COMUNICACIÓN ENTRE MÚLTIPLES VENTANAS.

En esta sección, vamos a ver cómo podemos comunicarnos con sub-ventanas abiertas empleando el método `open()` del objeto `window`.

Cada objeto `window` tiene una propiedad llamada `opener`. Esta propiedad contiene la referencia a la ventana o marco, que ha abierto ese objeto `window` empleando el método `open()`. Para la ventana principal el valor de `opener` será `null`.

Debido a que `opener` es una referencia válida a la ventana padre que abrió las otras, podemos emplearlo para iniciar la referencia a los objetos de la ventana original (padre) desde la ventana hija. Es semejante a lo que vimos con `frames`, pero en este caso es entre ventanas independientes del navegador.

3. OBJETOS NATIVOS EN JAVASCRIPT.

En esta sección vamos a echar una ojeada a los objetos nativos de JavaScript: aquellos disponibles para su utilización en cualquier aplicación. Estos objetos son: `String`, `Math`, `Number`, `Boolean` y `Date`.

3.1 OBJETO STRING.

Una cadena (string) consta de uno o más caracteres de texto, rodeados de comillas simples o dobles; da igual cuáles usemos ya que se considerará una cadena de todas formas, pero en algunos casos resulta más cómodo el uso de unas u otras. Por ejemplo, si queremos meter el siguiente texto dentro de una cadena de JavaScript,

```
<input  type="checkbox"  name="coche"  />Audi A6
```

podremos emplear tanto las comillas dobles o simples, tal y como se indica en el ejemplo:

```
var cadena = '<input  type="checkbox"  name="coche"  />Audi A6';
var cadena = "<input  type='checkbox'  name='coche'  />Audi A6";
```

Si queremos emplear comillas dobles tanto al principio y al final de la cadena como en el contenido, tendríamos que escaparlas con la barra invertida (\"), tal y como puedes observar en el siguiente ejemplo:

```
var cadena = "<input  type=\"checkbox\"  name=\"coche\"  />Audi A6";
```

Una de las operaciones más comunes a realizar con las cadenas es la de la concatenación. Anteriormente, ya has visto el operador que se emplea para realizar esta operación: la suma (+). Este operador ya lo hemos empleado para concatenar una cadena a un número o al contenido de una variable tal y como se muestra en el siguiente ejemplo:

```
var nombreEquipo  =  prompt("Introduce el nombre de tu equipo favorito:","");
var mensaje= "El  "  +  nombreEquipo +  "  ha sido el campeón de la Copa del Rey!";
alert(mensaje);
```

Hay veces en las que se quiere concatenar un texto muy largo porque se está construyendo el código (etiquetado) de una página y escribirlo todo en una línea no sería muy práctico a efectos de depuración. En este caso se suele emplear el operador += para concatenar en diferentes pasos un nuevo contenido al que ya tiene una variable. El siguiente ejemplo trata de ilustrar lo comentado en este párrafo:

```
var nuevoDoc = "";
nuevoDoc += '<!DOCTYPE html>';
nuevoDoc += '<html lang="es">' ;
nuevoDoc += ' <head>';
nuevoDoc += '  <meta http-equiv="content-type" content="text/html;charset=utf-8">';
```

El código anterior, aunque tiene cada etiqueta en una línea distinta, no quedaría de la misma forma cuando se ve el código fuente (suponiendo que de alguna forma ese contenido se enviara a una ventana recién creada). En realidad, quedaría todo en la misma línea. Para evitar esto, al igual que hacíamos con las comillas dobles en los valores de los atributos, utilizamos los caracteres especiales o caracteres de escape que ya vimos en la unidad anterior:

Caracteres de escape	
Caracteres	**Descripción**
\"	Comillas dobles
\'	Comilla simple
\\	Barra invertida
\b	Retroceso
\t	Tabulador
\n	Nueva línea
\r	Salto de línea
\f	Avance de página

TABLA 16 CARACTERES DE ESCAPE

3.1.1 Propiedades y Métodos del objeto String.

Para crear un objeto `String` lo podremos hacer de dos formas:

```
var miCadena = new String("texto de la cadena");
var miCadena = "texto de la cadena";
```

Esto quiere decir que por el hecho de tener una cadena de texto ya tenemos un objeto String con sus propiedades y sus métodos a los cuales podemos acceder o utilizar empleando la siguiente sintaxis:

```
cadena.propiedad;
cadena.metodo( [parámetros] );
```

Propiedades del objeto String	
Propiedad	**Descripción**
`length`	Contiene la longitud de la cadena de texto.

TABLA 17 PROPIEDADES DEL OBJETO STRING

Métodos del objeto String	
Método	**Descripción**
`charAt()`	Devuelve el carácter de la posición indicada entre paréntesis.
`charCodeAt()`	Devuelve el Unicode del carácter de la posición indicada entre paréntesis.
`concat()`	Devuelve el resultado de la unión (concatenación) de una o más cadenas.
`endsWith()`	Cheque si una cadena termina de una forma concreta.
`fromCharCode()`	Convierte valores Unicode a caracteres.
`includes()`	Chequea si una cadena contiene una subcadena concreta.
`indexOf()`	Devuelve la posición donde encuentra por primera vez el carácter buscado en la cadena.
`lastIndexOf()`	Devuelve la posición donde encuentra por última vez el carácter buscado en la cadena.
`localeCompare()`	a.localeCompare(b) Compara dos cadenas a y b devolviendo: • -1 si la cadena a es menor que b. • 0 si son iguales. • 1 si la cadena a es mayor que b.
`match()`	Devuelve las coincidencias encontradas entre una expresión regular y una cadena o null si no encuentra ninguna coincidencia.
`repeat()`	Devuelve una cadena con el resultado de la repetición veces de la cadena original.
`replace()`	Reemplaza una subcadena con una nueva cadena.
`search()`	Devuelve la posición dónde encontró una subcadena o la subcadena que concuerda con una expresión regular.
`slice()`	Devuelve una parte de la cadena.
`split()`	Divide una cadena en un array de subcadenas.
`startsWith()`	Chequea si una cadena comienza por algo concreto.
`substr()`	Devuelve un número determinado de caracteres comenzando en una determinada posición.
`substring()`	Extrae los caracteres de una cadena entre dos posiciones concretas.
`toLowerCase()`	Devuelve la cadena en minúsculas.
`toString()`	Devuelve el valor de un objeto String.
`toUpperCase()`	Devuelve la cadena en mayúsculas.
`trim()`	Quita los espacios sobrantes al principio y al final de una cadena.
`valueOf()`	Devuelve el valor primitivo de una cadena.

TABLA 18 MÉTODOS DEL OBJETO STRING

Todos los métodos retornan un nuevo valor. No modifican el valor original. Hay que hacer una reasignación.

Ejemplo: cadena=cadena.toUpperCase();

RECOMENDACIÓN

Observa lo que se visualiza por pantalla al ejecutar el siguiente ejemplo:

```
var cadena="El parapente es un deporte de riesgo medio";
document.write("La longitud de la cadena es: "+ cadena.length + "<br/>");
document.write(cadena.toLowerCase()+ "<br/>");
document.write(cadena.charAt(3)+ "<br/>");
document.write(cadena.indexOf('pente')+ "<br/>");
document.write(cadena.substring(3,16)+ "<br/>");
```

RECOMENDACIÓN 14 EJEMPLOS DE MÉTODOS DEL OBJETO STRING

También hay otros métodos que cambian el aspecto de una cadena. No son métodos estándar.

Métodos de presentación del objeto String

Método	Descripción
`anchor("nombre")`	Crea una etiqueta ancla `<a name="nombre">cadena</a>` donde `cadena` es el objeto `String` que llama al método `anchor`.
`big()`	Visualiza la cadena utilizando una fuente grande.
`blink()`	Visualiza la cadena parpadeando.
`bold()`	Visualiza la cadena en negrita.
`fixed()`	Visualiza la cadena utilizando una fuente regular.
`fontcolor("color")`	Visualiza la cadena utilizando un color concreto.
`fontsize(tamaño)`	Visualiza la cadena utilizando un tamaño concreto.
`italics()`	Visualiza la cadena en cursiva.
`link("url")`	Visualiza la cadena creando un hiperenlace.
`small()`	Visualiza la cadena utilizando una fuente pequeña.
`strike()`	Visualiza la cadena tachada.
`sub()`	Visualiza la cadena como subíndice.
`sup()`	Visualiza la cadena como superíndice.

TABLA 19 PROPIEDADES DE PRESENTACIÓN DEL OBJETO STRING

RECOMENDACIÓN

En el siguiente enlace puedes comprobar el funcionamiento de las propiedades y métodos del objeto String.

Objeto String

RECOMENDACIÓN 15 PROPIEDADES Y MÉTODOS DEL OBJETO STRING

EJERCICIOS (del boletín).

Escribe el código de los archivos necesarios (html y js) para realizar un programa que solicite la entrada de un nombre (con un prompt) y muestre el resultado de la ejecución con un único alert:

I. Introducir un nombre completo por teclado con el formato: ***apellidos, nombre***
   El programa debe avisar del posible error encontrado:
   a. Se dejó en blanco el nombre o los apellidos
   b. No hay la coma de separación o a hay más de una.

II. Depurar la escritura del nombre (eliminando espacios sobrantes a la izquierda, derecha y por el medio).
   Antes de la coma no debe haber blanco y después de la coma debe haber un espacio.

III. Visualizar el nombre antes y después de la depuración.

IV. Visualizar un saludo personalizado por pantalla (Ejemplo: Hola José, para el nombre Pérez López, José Luis).

V. Visualizar la longitud del nombre completo ya depurado (la coma no se tiene en cuenta).

VI. Visualizar el nombre depurado en minúsculas.

VII. Visualizar el nombre depurado en mayúsculas.

VIII. Visualizar la longitud del nombre propio ya depurado.

IX. Visualizar la longitud de sus apellidos ya depurados.

X. Visualizar las iniciales seguidas de punto.
   Primero las iniciales del nombre y después las de los apellidos.

XI. Encriptar el nombre y sus apellidos depurados con un * por cada carácter.
   La coma no se encripta ni se pone.
   Los espacios en blanco no se encriptan pero se ponen.
   Visualizar el resultado de la encriptación.

XII. Visualizar si el conjunto de sus iniciales (del apartado X) forman una palabra palíndroma.
   Si se lee de la misma forma de izquierda a derecha que de derecha a izquierda.
   No se tienen en cuenta los puntos que siguen a cada inicial.

IMPORTANTE:
Para hacer este ejercicio se puede emplear las propiedades y métodos de todos los objetos vistos hasta el momento, aunque se puede resolver empleando únicamente métodos del objeto string o métodos de arrays.
Debe de funcionar cualquiera que sea el nombre introducido.

En la siguiente tabla se muestra un ejemplo de la salida (solo el dato) que debería mostrar en cada apartado del ejercicio si el dato introducido por el usuario fuera **" Gómez López , Luis Gabriel "** (sin las comillas):

Apartado	
III.	" Gómez López , Luis Gabriel "
III.	"Gómez López, Luis Gabriel"
IV.	Hola Luis
V.	24
VI.	gómez lópez, luis gabriel
VII.	GÓMEZ LÓPEZ, LUIS GABRIEL
VIII.	12
IX.	11
X.	L.G.G.L.
XI.	**** ******* ***** *****
XII.	Sí

Añade comentarios al código (variables globales, bloques de código, prototipo de funciones, operaciones especiales).
Tabula de forma correcta.
Estructura correctamente el código:
Funciones-Variables globales-Código,
Variables globales-Código-Funciones,
Variables globales-Funciones-Código

EJERCICIO 1 OBJETO STRING

3.2 OBJETO MATH.

Ya vimos anteriormente algunas funciones, que nos permitían convertir cadenas a diferentes formatos numéricos (`parseInt`, `parseFloat`). Aparte de esas funciones, disponemos de un objeto `Math` en JavaScript, que nos permite realizar operaciones matemáticas. El objeto `Math` no es un constructor (no nos permitirá por lo tanto crear o instanciar nuevos objetos que sean de tipo `Math`), por lo que, para llamar a sus propiedades y métodos, lo haremos anteponiendo `Math` a la propiedad o el método. Por ejemplo:

```
var x = Math.PI;            // Devuelve el número PI.
var y = Math.sqrt(16);      // Devuelve la raíz cuadrada de 16.
```

Propiedades del objeto Math	
Propiedad	**Descripción (son constantes)**
`E`	Devuelve el número Euler o constante de Napier (aproximadamente 2,718).
`LN2`	Devuelve el logaritmo neperiano de 2 (aproximadamente 0,693).
`LN10`	Devuelve el logaritmo neperiano de 10 (aproximadamente 2,302).
`LOG2E`	Devuelve el logaritmo en base 2 del número E (aproximadamente 1,442).
`LOG10E`	Devuelve el logaritmo en base 10 del número E (aproximadamente 0,434).
`PI`	Devuelve el número PI (aproximadamente 3,14159).
`SQRT1_2`	Devuelve la raíz cuadrada de ½ (aproximadamente 0,707106).
`SQRT2`	Devuelve la raíz cuadrada de 2 (aproximadamente 1,41421).

TABLA 20 PROPIEDADES DEL OBJETO MATH

Métodos del objeto Math	
Método	**Descripción**
`abs(x)`	Devuelve el valor absoluto de x.
`acos(x)`	Devuelve el arcocoseno de x, en radianes.
`asin(x)`	Devuelve el arcoseno de x, en radianes.
`atan(x)`	Devuelve el arcotangente de x, en radianes con un valor entre -PI/2 y PI/2.
`atan2(y,x)`	Devuelve el arcotangente del cociente de sus argumentos.
`ceil(x)`	Devuelve el número x redondeado al alza al entero más próximo.
`cos(x)`	Devuelve el coseno de x (x está en radianes).
`exp(x)`	Devuelve E elevado a x.
`floor(x)`	Devuelve el número x redondeado a la baja al entero más próximo.
`log(x)`	Devuelve el logaritmo neperiano (en base E) de x.
`max(x,y,z,...,n)`	Devuelve el número más alto de los que se pasan como parámetros.
`min(x,y,z,...,n)`	Devuelve el número más bajo de los que se pasan como parámetros.
`pow(x,y)`	Devuelve el resultado de x elevado a y.
`random()`	Devuelve un número al azar entre 0 y 1.
`round(x)`	Redondea x al entero más próximo.
`sin(x)`	Devuelve el seno de x (x está en radianes).
`sqrt(x)`	Devuelve la raíz cuadrada de x.
`tan(x)`	Devuelve la tangente de un ángulo.
`trunc(x)`	Devuelve la parte entera de x.

TABLA 21 MÉTODOS DEL OBJETO MATH

RECOMENDACIÓN

En el siguiente enlace puedes comprobar el funcionamiento de las propiedades y métodos del objeto Math.

Objeto Math

RECOMENDACIÓN 16 PROPIEDADES Y MÉTODOS DEL OBJETO MATH

EJERCICIO

- Haz una función que permita generar una lista de N números enteros aleatorios distintos comprendidos entre 2 números concretos.
- Haz un programa que pida el tipo de juego al que quieres jugar (Euromillón, Primitiva, Bonoloto, El Gordo), haciendo uso de la función anterior, muestre con un alert la combinación que juegas (ordenada).

  En cada juego varían la cantidad de números a seleccionar, así como el rango de números.
  Algunos tienen números adicionales (estrellas).
  La función pedida tiene que valer para todos ellos.

  Primitiva- 6 números [1,49] + 1 reintegro [0,9]
  El Gordo- 5 números [1,54] + 1 reintegro [0,9]
  Euromillón- 5 números [1,50] + 2 estrellas [1,12]
  Bonoloto- 6 números [1,49]

EJERCICIO 2 OBJETO MATH

```
document.write(Math.cos(3) + "<br />");

document.write(Math.asin(0) + "<br />");

document.write(Math.max(0,150,30,20,38) + "<br />");

document.write(Math.pow(7,2) + "<br />");

document.write(Math.round(0.49) + "<br />");
```

EJEMPLO 5 OBJETO MATH

3.3 OBJETO NUMBER.

El objeto `Number` se usa muy raramente, ya que, para la mayor parte de los casos, JavaScript satisface las necesidades del día a día con los valores numéricos que almacenamos en variables. Pero el objeto `Number` contiene alguna información y capacidades muy interesantes para programadores más serios.

Curiosidades del objeto Number.

`NaN, isNaN(x)`

Las operaciones multiplicación (*), división (/) y resta (-) funcionan igualmente, aunque los números estén escritos entre comillas.

Eso no ocurre con la suma (+) ya que este operador funciona también concatenando cadenas.

`NaN` (Not a Number) es una palabra reservada para indicar que un número ha intentado almacenar algo que no es realmente un número.

La función global `isNaN(x)` permite averiguar si el número x es realmente un `NaN`.

```
var x=10;
var y="Hola";
var z=x*y;
if (isNaN(z)) {
    //z=NaN (ya que se ha intentado multiplicar 10 por "Hola").
    //por lo tanto isNaN(z) devuelve true
    alert(typeof z); // dice "number" porque NaN es un number
} else {
    //z es un número de verdad con el que se puede hacer cualquier operación
}
```

`Infinity, -Infinity`

Son valores que JavaScript devuelve cuando se sobrepasa los valores máximos o mínimos de los números en alguna operación.

```
var x=2/0; // x = Infinity
var x=-2/0 // x = -Infinity
alert(typeof x); // dice "number"
```

Hexadecimales

Cualquier constante numérica precedida de 0x es interpretada como un número hexadecimal.

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

Un número y un objeto Number no son exactamente iguales aunque contengan el mismo valor.

```
var x=500;
var y=new Number(500);
//(x==y) es cierto
//(x===y) es falso porque typeof(x)=number, mientras que typeof(y)=object

//y además
var z=new Number(500);
//(y===z) es falso porque a pesar de ser los 2 objetos, los objetos no pueden compararse.
```

El objeto `Number` contiene propiedades que nos indican el rango de números soportados en el lenguaje. El número más alto es 1.79.....E + 308; el número más bajo es 5E-324. Cualquier número mayor que el número más alto, será considerado como infinito positivo, y si es más pequeño que el número más bajo, será considerado infinito negativo.

Los números y sus valores están definidos internamente en JavaScript, como valores de doble precisión y de 64 bits.

El objeto `Number`, es un objeto envoltorio para valores numéricos primitivos.

Los objetos `Number` son creados con `new Number()`.

Propiedades del objeto Number	
Propiedad	**Descripción**
`constructor`	Devuelve la función que creó el objeto `Number`.
`MAX_VALUE`	Devuelve el número más alto disponible en JavaScript.
`MIN_VALUE`	Devuelve el número más pequeño disponible en JavaScript.
`NEGATIVE_INFINITY`	Representa a infinito negativo (se devuelve en caso de overflow).
`NaN`	Representa un valor "Not a Number"
`POSITIVE_INFINITY`	Representa a infinito positivo (se devuelve en caso de overflow).
`prototype`	Permite añadir nuestras propias propiedades y métodos a un objeto.

TABLA 22 PROPIEDADES DEL OBJETO NUMBER

Métodos del objeto Number	
Método	**Descripción**
`isFinite()`	Chequea si el número es un valor finito.
`isInteger()`	Chequea si es un valor entero.
`isNaN()`	Chequea si es un "Not a Number".
`isSafeInteger()`	Chequea si un número es un entero en el rango $[-(2^{53} - 1) , (2^{53} -1)]$.
`toExponential(x)`	Convierte un número a su notación exponencial.
`toFixed(x)`	Formatea un número con `x` digitos decimales después del punto decimal.
`toPrecision(x)`	Formatea un número a `x` dígitos de longitud. (Sin incluir el punto decimal)
`toString(x)`	Convierte un objeto `Number` en una cadena. (x debe ser un número comprendido entre 2 y 36, será la base a la que se quiere cambiar el nº). • Si x=2 se mostrará el número en binario. • Si x=8 se mostrará el número en octal. • Si x=16 se mostrará el número en hexadecimal. • Si x se omite mostrará el mismo número.
`valueOf()`	Devuelve el valor primitivo de un objeto `Number`.

TABLA 23 MÉTODOS DEL OBJETO NUMBER

Los métodos del objeto Number devuelven un nuevo valor. No modifican el original.

RECOMENDACIÓN

En el siguiente enlace puedes comprobar el funcionamiento de las propiedades y métodos del objeto Number.

Objeto Number

RECOMENDACIÓN 17 PROPIEDADES Y MÉTODOS DEL OBJETO NUMBER

```
//Comprueba los valores que devuelven
var num = new Number(13.3714);

alert(document.write(num.toPrecision(3));

alert(num.toFixed(1)+);

alert(num.toString(2)+);

alert(num.toString(8)+);

alert(num.toString(16)+);

alert(Number.MIN_VALUE);

alert(Number.MAX_VALUE);
```

EJEMPLO 6 OBJETO NUMBER

3.4 OBJETO BOOLEAN.

El objeto `Boolean` se utiliza para convertir un valor no Booleano a un valor Booleano (true o false).

Propiedades del objeto Boolean	
Propiedad	**Descripción**
`constructor`	Devuelve la función que creó el objeto `Boolean`.
`prototype`	Permite añadir nuestras propias propiedades y métodos a un objeto.

TABLA 24 PROPIEDADES DEL OBJETO BOOLEAN

Métodos del objeto Boolean	
Método	**Descripción**
`toString(x)`	Convierte un valor `Boolean` en una cadena y devuelve el resultado (`true` o `false`).
`valueOf()`	Devuelve el valor primitivo de un objeto `Boolean`.

TABLA 25 MÉTODOS DEL OBJETO BOOLEAN

```
var bool = new Boolean(1);

document.write(bool.toString());

document.write(bool.valueOf());
```

EJEMPLO 7 OBJETO BOOLEAN

Se puede utilizar la función Boolean para saber si el resultado de una expresión es cierta o no. A continuación tienes una serie de ejemplos:

```
// todas las variables b1...b7 valdrán true
var b1 = new Boolean(100);
var b2 = new Boolean(3.14);
var b3 = new Boolean(-15);
var b4 = new Boolean("Hello");
var b5 = new Boolean('false');
var b6 = new Boolean(1 + 7 + 3.14);
var b7 = new Boolean(true);

//todas las variables a1..a7 valdrán false
var a1 = new Boolean(0);
var a2 = new Boolean(-0);
var a3 = new Boolean("");
var a4;    //es Undefined
alert(new Boolean(a4));
var a5 = new Boolean(null);
var a6 = new Boolean(false);
var a7 = new Boolean(NaN);

//comparando
var x = 0;
var y = new Boolean(0);
alert(x == y);    //devuelve true porque los 2 son false
alert(x === y);   //devuelve false porque y es un objeto y x un número.

//recuerda que los objetos no se pueden comparar.
```

EJEMPLO 8 EJEMPLOS CON LA FUNCIÓN BOOLEAN()

3.5 OBJETO DATE.

El objeto `Date` se utiliza para trabajar con fechas y horas.

Una fecha en Java Script se puede escribir:

Sat Dec 09 2017 10:05:06 GMT+0100

o bien en número:

1512810306191

Cuando se escribe en número representa el número de milisegundos que han pasado desde el 1 de enero de 1970 a las 00:00:00.

Un día tiene 24 horas, un total de 86.400.000 milisegundos (24x60x60x1000).

Los objetos `Date` se crean con `new Date()`.

Hay 4 formas de instanciar (crear) un objeto de tipo `Date`:

```
var d = new Date();                          //la fecha y hora actuales

//var d = new Date(milisegundos);
var d = new Date(1512810306191);             //la fecha mostrada al principio

//var d = new Date(cadena_de_Fecha);
var d = new Date("October 13, 2014 11:13:00");

//var d = new Date(año, mes, día, horas, minutos, segundos, milisegundos);
var d = new Date(2017,11,9,10,20,0,0);
// (el mes comienza en 0, Enero sería 0, Febrero 1, etc.)

//especificando solo año mes y día (la hora se supone todo 0)
var d = new Date(2017,11,9);
```

EJEMPLO 9 FORMAS DE INSTANCIAR EL OBJETO DATE

Propiedades del objeto Date	
Propiedad	**Descripción**
`constructor`	Devuelve la función que creó el objeto `Date`.
`prototype`	Permite añadir nuestras propias propiedades y métodos a un objeto.

TABLA 26 PROPIEDADES DEL OBJETO DATE

Cuando se visualiza una fecha se visualiza como cadena:

```
var d = new Date();                          //la fecha y hora actuales
alert(d);
alert(d.toString());
//ambos devuelven el mismo resultado que en el momento de la ejecución es:
// Sat Dec 09 2017 10:29:41 GMT+0100

alert(d.toUTCString());
//para visualizar la fecha en un formato más como el nuestro
//fijaros que no hace el cambio horario
// Sat, 09 Dec 2017 09:29:41 GMT
```

EJEMPLO 10 VISUALIZAR EL OBJETO DATE

Métodos del objeto Date	
Método	**Descripción**
`getDate()` `setDate()`	Devuelve o establece el día del mes (de 1-31).
`getDay()`	Devuelve el día de la semana (de 0-6). (0-Domingo,….6-Sábado)
`getFullYear()` `setFullYear()`	Devuelve o establece el año (4 dígitos).
`getHours()` `setHours()`	Devuelve o establece la hora (de 0-23).
`getMilliseconds()` `setMilliseconds()`	Devuelve o establece los milisegundos (de 0-999).
`getMinutes()` `setMinutes()`	Devuelve o establece los minutos (de 0-59).
`getMonth()` `setMonth()`	Devuelve o establece el mes (de 0-11).
`getSeconds()` `setSeconds()`	Devuelve o establece los segundos (de 0-59).
`getTime()` `setTime()`	Devuelve o establece los milisegundos desde media noche del 1 de Enero de 1970.
`getTimezoneOffset()`	Devuelve la diferencia de tiempo entre GMT y la hora local en minutos.
`getUTCDate()` `setUTCDate()`	Devuelve o establece el día del mes en base a la hora UTC (de 1-31).
`getUTCDay()`	Devuelve el día de la semana en base a la hora UTC (de 0-6).
`getUTCFullYear()` `setUTCFullYear()`	Devuelve o establece el año en base a la hora UTC (4 dígitos).
`getUTCHours()` `setUTCHours()`	Como getHours() pero hora UTC.
`getUTCMilliseconds()` `setUTCMilliseconds()`	Como getMilliseconds() pero milisegundos UTC.
`getUTCMinutes()` `setUTCMinutes()`	Como getMinutes() pero minutos UTC.
`getUTCMonth()` `setUTCMonth()`	Como getMonth() pero mes UTC.
`getUTCSeconds()` `setUTCSeconds()`	Como getSeconds() pero segundos UTC.
`now()`	Número de milisegundos desde el 1 de enero de 1970. (Date.now())
`parse()`	Parsea una cadena de fecha y devuelve el número de milisegundos desde el 1 de enero de 1970. (Date.parse("Aug 9, 1995"))
`toDateString()`	Convierte la fecha de un objeto Date en un string.
`toISOString()`	Covierte la fecha a string utilizando el estándar ISO.
`toJSON()`	Covierte la fecha a string a fecha con formato JSON.
`toLocaleDateString()`	Convierte la fecha de un objeto Date en un string usando la convención local.
`toLocaleTimeString()`	Convierte la hora de un objeto Date en un string usando la convención local.
`toLocaleString()`	Convierte un objeto Date en un string usando la convención local.
`toString()`	Convierte un objeto Date en un string.
`toTimeString()`	Convierte la hora de un objeto Date en un string.
`toUTCString()`	Convierte un objeto Date en un string de acuerdo al tiempo universal.
`UTC()`	Devuelve el número de milisegundos desde la medianoche del 1 de enero de 1970 de acuerdo a la hora UTC. (Date.UTC(año,mes[, dia[, hora[, minutos[, segundos, milisegundos]]]])
`valueOf()`	Retorna el valor primitivo de un objeto Date.

TABLA 27 MÉTODOS DEL OBJETO DATE

UTC – Tiempo (Hora) Universal Coordinado.
GMT – Tiempo (Hora) Media Greenwich.

RECOMENDACIÓN

En el siguiente enlace puedes comprobar el funcionamiento de las propiedades y métodos del objeto Date.

Objeto Date

RECOMENDACIÓN 18 PROPIEDADES Y MÉTODOS DEL OBJETO DATE

```
var d = new Date();
alert(d.toLocaleString());    // muestra 9/12/2017 20:22:17
alert(d.toISOString());       // muestra 2017-12-09T19:23:12.909Z
alert(d.toJSON());            // lo mismo
//la fecha JSON tiene el mismo formato que el estándar ISO-8601:
//YYYY-MM-DDTHH:mm:ss.sssZ  (Z significa Zero de desplazamiento con respecto al tiempo UTC
```

EJEMPLO 11 OBJETO DATE

EJERCICIOS

- Haz un programa que diga sabiendo el día en que naciste nos diga en qué día de la semana (lunes, martes, ...) caerá tu próximo cumpleaños.

- Completa el ejercicio anterior para que pueda decir el día del próximo cumpleaños de cualquier fecha de nacimiento introducida por teclado con el formato dd-mm-aaaa

EJERCICIO 3 OBJETO DATE

4. EXPRESIONES REGULARES Y OBJETOS REGEXP.

Las expresiones regulares son patrones de búsqueda que se pueden utilizar para encontrar texto que coincida con dicho patrón.

```
var texto = "Los trenes de hoy en día alcanzan grandes velocidades";
var subcadena = "velocidad";
var i = texto.indexOf(subcadena);
// devuelve 42, posición donde encontró la subcadena dentro del texto

if (i != -1) // correcto, se ha encontrado la subcadena
   alert ("Encontrado en la posición"+(i+1)); //la primera posición es la 0
```

EJEMPLO 12 USANDO MÉTODOS DEL OBJETO STRING

El ejemplo anterior encuentra "velocidad" aunque no es una palabra completa sino el comienzo de la palabra velocidades. ¿Y si quisiéramos que el código sólo nos dijera la posición donde encontró la palabra en el caso de que esta sea una palabra completa? El código debería comprobar que después de la última letra "d" de la palabra "velocidad" hay un espacio en blanco o bien algún símbolo de puntuación (".", ",", ":", ";", "?", "!", ")") o bien un espacio en blanco o el símbolo de comillas dobles o simples, que garantice que es una palabra completa. Esto implica un conjunto de condicionales a mayores que prolongan innecesariamente el código. La mejor solución es usar una expresión regular.

En JavaScript las expresiones regulares se gestionan a través del objeto **`RegExp`**.

Para crear un literal del tipo **`RegExp`** tendrás que usar la siguiente sintaxis:

```
var  expresion = /expresión_regular/[flags];
```

EJEMPLO 13 SINTAXIS DE LA CREACIÓN DE UNA EXPRESIÓN REGULAR

La expresión regular está contenida entre dos barras (/). Fíjate que no lleva comillas. Las comillas sólo se pondrán en la expresión regular cuando formen parte del patrón en sí mismo.

Las expresiones regulares se construyen con caracteres que podrán ir solos o en combinación de caracteres especiales.

El siguiente ejemplo es una expresión regular que establece un patrón que permitiría realizar una búsqueda de las palabras que contengan las palabras "Aloe" y "Vera", colocadas en ese orden y separadas por uno o más espacios en blanco en el medio:

```
var expresion=/Aloe\s+Vera/;
```

EJEMPLO 14 SINTAXIS DE LA CREACIÓN DE UNA EXPRESIÓN REGULAR

Los caracteres especiales en este ejemplo son:

- La barra invertida (\), que tiene puede tener dos efectos:
  - si se utiliza seguida de un carácter regular indica que éste se trata de un carácter especial,
  - se usa con un carácter especial como el signo más (+) indica que el carácter (\) debe ser tratado literalmente.

  En este caso, la barra invertida se utiliza seguida de una "s" por lo que esta letra pasa a ser un carácter especial que indica un espacio en blanco.
- El símbolo de suma (+) indica que el carácter anterior (\s) puede aparecer una o más veces.

4.1 CARACTERES ESPECIALES Y FLAGS EN EXPRESIONES REGULARES.

Veamos una tabla con algunos de los caracteres más utilizados para la construcción de Expresiones Regulares:

Caracteres especiales utilizados en expresiones regulares			
Carácter	**Coincidencias**	**Patrón**	**Ejemplo de cadena**
`.`	Cualquier carácter excepto nueva línea	`/a.e/`	Que aparezca cualquier carácter, excepto nueva línea entre la a y la e: "ape" y "axe"
`\W`	Coincide con caracteres que **NO** sean (letras, dígitos, subrayados). El espacio en blanco no es considerado letra.	`/\W/`	Que aparezca un carácter (que no sea letra, dígito o subrayado): "%" en "100%"
`\w`	Coincide con caracteres del tipo (letras, dígitos, subrayados)	`/\w/`	Que aparezca un carácter (letra, dígito o subrayado): "J" en "JavaScript". No en "?!%"
`\D`	Cualquier carácter que **NO** sea un dígito	`/\D{2,4}/`	Que aparezcan mínimo 2 y máximo 4 caracteres que no sean dígitos: encontrará la cadena "Ahor" en "Tienes Ahora 45 años"
`\d`	Dígitos del 0 al 9	`/\d{3}/`	Que aparezcan exactamente 3 dígitos: encontrará la cadena "456" en "Ahora en 456"
`\B`	Coincide al final de una palabra	`/\Bno/`	Que "no" esté al final de una palabra: "este invierno" ("no" de "invierno")
	No coincide con el comienzo ni con el final de la palabra	`/\Bno\B/`	Palabras que contenga "no": renovado
`\b`	Coincide con el inicio de una palabra	`/\bno/`	Que "no" esté al comienzo de una palabra: "novedad"
	Coincide con el comienzo y el final de la palabra	`/\bno\b/`	La palabra "no" en "puede que no vaya". No coincidiría en "No hay novedades"
`[^…]`	Cualquier carácter excepto los que están entre corchetes	`/a[^px]e/`	Que aparezca cualquier carácter excepto la "p" o la "x" después de la letra a y antes de la letra e: "ale", pero no "axe" o "ape"
`[…]`	Cualquier carácter entre corchetes	`/a[px]e/`	Que aparezca alguno de los caracteres "p" o "x" entre la a y la e: "ape", "axe", pero no "ale"
`^`	Al inicio de una cadena	`/^Esto/`	Coincidencia en "Esto es..."
`$`	Al final de la cadena	`/final$/`	Coincidencia en "Esto es el final"
`*`	Coincide 0 o más veces	`/se*/`	Que la "e" aparezca 0 o más veces después de una letra s: "seeee" y también "se"
`?`	Coincide 0 o 1 vez	`/ap?/`	Que la p aparezca 0 o 1 vez después de la letra a: "apple" y "and"
`+`	Coincide 1 o más veces	`/ap+/`	Que la "p" aparezca 1 o más veces después de la letra a: "apple" pero no "and"
`{n}`	Coincide exactamente n veces	`/ap{2}/`	Que la "p" aparezca exactamente 2 veces después de la letra a: "apple" pero no "apabullante"
`{n,}`	Coincide n o más veces	`/ap{2,}/`	Que la "p" aparezca 2 o más veces después de la letra a: "apple" y "appple" pero no en "apabullante"
`{n,m}`	Coincide al menos n, y máximo m veces	`/ap{2,4}/`	Que la "p" aparezca al menos 2 veces y como máximo 4 veces después de la letra a: "a**pppp**pple" (encontrará 4 "p")
`p1|p2`	Coincide con p1 (palabra 1) o p2 (palabra 2)	`/tu|mi/`	Contiene la palabra tu o mi: Pedro es mi nombre
`(…)`	Agrupa caracteres	`/(va)$/`	Termina en la agrupación de letras "va": renueva
`\n`	Coincide con una nueva línea		
`\s`	Coincide con un espacio en blanco		
`\S`	Coincide con un carácter que **NO** es un espacio en blanco		
`\t`	Un tabulador		
`\r`	Un retorno de carro		

TABLA 28 CARACTERES ESPECIALES UTILIZADOS EN EXPRESIONES REGULARES

Flags o indicadores de las expresiones regulares

Flag	Significado
`g`	Coincidencia global: comprueba en toda la cadena, en lugar de detenerse cuando encuentra la primera coincidencia.
`i`	No es sensible a mayúsculas y/o minúsculas.
`m`	Se aplica el carácter especial de comenzar y terminar la línea (^ y $, respectivamente) a cada línea en una cadena de varias líneas. Para que lo haga en más de una línea hay que indicar también el flag g.

TABLA 29 FLAGS O INDICADORES EN LAS EXPRESIONES REGULARES

RECOMENDACIÓN

En los siguientes enlaces podrás ver otros ejemplos de expresiones regulares y alguna información adicional relacionada con este tema.

Enlace 1 Enlace 2

RECOMENDACIÓN 19 EXPRESIONES REGULARES

4.2 EL OBJETO REGEXP.

El objeto **RegExp** es tanto un literal como un objeto de JavaScript, por lo que también se podrá crear usando un constructor:

```
var expresionregular = new RegExp("Texto Expresión Regular");
```

EJEMPLO 15 CONSTRUCTOR DE UNA EXPRESIÓN REGULAR

¿Cuándo usar el literal o el objeto?

La expresión **RegExp** literal es compilada cuando se ejecuta el script, por lo tanto, se recomienda usar el literal cuando sabemos que la expresión no cambiará. Una versión compilada es mucho más eficiente.

Usaremos el objeto, cuando sabemos que la expresión regular va a cambiar, o cuando vamos a proporcionarla en tiempo de ejecución.

Al igual que otros objetos en JavaScript, el objeto **RegExp** también tiene sus propiedades y métodos:

Propiedades del objeto RegExp	
Propiedad	**Descripción**
`global`	Especifica que se utilizará el modificador "g".
`ignoreCase`	Especifica que se utilizará el modificador "i".
`lastIndex`	Índice donde comenzará la siguiente búsqueda.
`multiline`	Especifica si el modificador "m" es utilizado.
`source`	El texto de la expresión regular RegExp.

TABLA 30 PROPIEDADES DEL OBJETO REGEXP

Métodos del objeto RegExp	
Método	**Descripción**
`compile()`	Compila una expresión regular.
`exec()`	Busca la coincidencia en una cadena. Devolverá la primera coincidencia.
`test()`	Busca la coincidencia en una cadena. Devolverá true o false.

TABLA 31 MÉTODOS DEL OBJETO REGEXP

Ejemplos de uso de expresiones regulares:

Para comprobar si una cadena está incluida (contenida) en otra.

```
var datos = new Array();
datos[0] = "El Blogger de Google"; // aquí dará verdadero
datos[1] = "El blogger de Google"; // aquí dará falso (por la mayúscula B)
datos[2] = "BloggerGoogle";        // aquí dará verdadero
datos[3] = "Google Blogger";       // aquí dará falso (por el orden en el que están)
var patron = /Blog.*Goog/;  // Patrón de búsqueda que contenga en cualquier posición:
                            // Blog -la cadena "Blog"
                            // . -seguido de cualquier carácter excepto la nueva línea
                            // * -el carácter anterior 0 o más veces
                            // Goog -seguido de la cadena "Goog"
for (var i = 0; i < datos.length; i++)
  alert(datos[i] + " " + patron.test(datos[i]));
```

EJEMPLO 16 EXPRESIONES REGULARES

Validación de un número de Seguridad Social Americano.

Un número de Seguridad Social americano consiste en 8 dígitos agrupados en tres campos separados, generalmente, por guiones: AAA-GG-SSS.

- Los tres primeros dígitos (AAA), corresponden al "Número de área".
- Los dos siguientes (GG), corresponden al "Número de grupo".
- Los 3 últimos (SSS), corresponden al "Número de serie".

En este ejemplo se valida que un número facilitado cumpla el formato indicado con o sin guiones ya que éstos son opcionales (caracteres "-?" dentro de la expresión regular).


```
function esCorrectoNSSAmericano(numero){
   var patron = /^\d{3}-?\d{2}-?\d{3}$/;
   patron.test(numero)?return true;:return false;
}


var unNumero=prompt(Introduce un n° de la seguridad social con formato americano\n
AAA-GG-SSS);
if (esCorrectoNSSAmericano(unNumero){
  alert("Correcto: el número "+numero+" cumple el estándar americano");
} else {
  alert("Error: el número "+numero+" NO cumple el estandar.");
}
```

Ejemplo 17 Expresiones regulares. Validación de un número de Seguridad Social Americano.

RECOMENDACIÓN

Trata de hacer los ejercicios anteriores empleando expresiones regulares para la validación de datos de entrada:

- En el ejercicio de strings para comprobar que al menos hay una letra antes de la coma y otra después.
- En el ejercicio de loterías para comprobar que la opción escogida pertenece a uno de los juegos.
- En el ejercicio del próximo cumpleaños para comprobar que la fecha de nacimiento introducida tiene el formato correcto dd-mm-aaaa pudiendo introducirse fechas comprendidas entre el 1 de enero de 1960 y la fecha actual. (También debe permitir el formato d-m-aaaa)

RECOMENDACIÓN 20 MEJORAS EJERCICIOS

GLOSARIO

Binario. Es un sistema numérico en base 2(usa 2 símbolos) y utiliza los dígitos 0 y 1 como representación numérica.

Hexadecimal. Es un sistema numérico en base 16(usa 16 símbolos) y utiliza los dígitos del 0 a 9 y las letras de la A a la F como representación numérica.

Octal. Es un sistema numérico en base 8(usa 8 símbolos) y utiliza los dígitos del 0 al 7 como representación numérica.

Unicode. Es un estándar de codificación de caracteres diseñado para facilitar el tratamiento informático, transmisión y visualización correcta de textos realizados en diferentes idiomas, de tal forma que los caracteres especiales de cada idioma sean mostrados correctamente independientemente de nuestra configuración de idioma local. El término Unicode proviene de los tres objetivos perseguidos: universalidad, uniformidad y unicidad.