Unidad 3:

Modelo de objetos predefinidos en javascript

Contenido

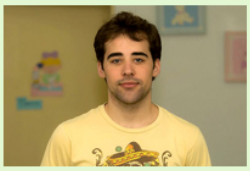
Caso práctico

Antonio sabe que no es suficiente para poder desarrollar una web o aplicación mínimamente decente y decide tener una breve conversación con **Ada** y posteriormente decide reunirse con tutor **Juan** con el cual discute sus progresos hasta el momento.

Como **Antonio** ha alcanzado satisfactoriamente los objetivos de la unidad de introducción y ya sabe los fundamentos del lenguaje de programación **JavaScript** decide que ya es el momento de aprender los objetos, que no las clases, que le van a permitir gestionar ventanas, marcos, propiedades de los navegadores, de las URL, cadenas de texto, matemáticas, fechas, etc. en **JavaScript**.

1.- Objetos de más alto nivel en javascript.

Caso práctico



Antonio, se va a enfrentar a la base de objetos que se utilizan en JavaScript. Con su tutor **Juan** no le cabe ninguna duda de que aprenderá a utilizar los objetos que se utilizan el prácticamente todos los scripts y aplicaciones utilizadas en la Web. No sólo podrá abrir nuevas ventanas, y saber que página está abierta, o el puerto por donde se ha conectado; también podrá averiguar con que sistema operativo y navegador se han conectado a su web entre otras cosas muy interesantes relacionadas con los enlaces e imágenes que se encuentran en el documento.

Comprender estos objetos con su propiedades y métodos le va a permitir a Antonio desarrollar webs y aplicaciones que desarrolle, no solo en este módulo, sino en su futuro. No debe despistarse.

¿Cómo averiguar que navegador estáis utilizando? Es una pregunta recurrente, ya hay diferencias entre los distintos navegadores. Cada navegador dispone de distintos métodos y propiedades, o sea, no todos los navegadores hacen lo mismo de la misma forma. Pero no hay que asustarse. Para evitar este tipo de problemas se puede utilizar el subconjunto definido en ECMAScript. Ya podemos disponer de unos métodos y propiedades que en teoría nos van a dar menos problemas. E incluso Microsoft los soporta. Pero, siempre hay un pero, dentro de lo que es estándar puede incluso darse diferencias sutiles a la hora de implementar, o sea, de como cada motor JavaScript trabaja con cada objeto o bugs de los mismos. Son pocos, pero seguramente veréis algo extraño alguna vez.

En resumidas cuentas, leer, comprender y aplicar los conocimientos en éste u otros navegadores y prestar atención a que propiedades y métodos deberéis utilizar.

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, sobre 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 la unidad anterior, los fundamentos de la programación, vamos a profundizar un poco más en lo que se refiere a los objetos, que podremos colocar en la mayoría de nuestros documentos.

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

La forma de acceder a una propiedad de un objeto es la siguiente:

```
nombreobjeto.propiedad
```

La forma de acceder a un método de un objeto es la siguiente:

```
nombreobjeto.metodo( [parámetros opcionales] )
```

También podemos referenciar a una propiedad de un objeto, por su índice en la creación. Los índices comienzan por 0.

En esta unidad, nos enfocaremos en objetos de alto nivel, que encontrarás frecuentemente en tus aplicaciones de JavaScript: `window`, `location`, `navigator` y `document`. El objetivo, no es solamente indicarte las nociones básicas para que puedas comenzar a realizar tareas sencillas, sino también, el prepararte para profundizar en las propiedades y métodos, gestores de eventos, etc. que encontrarás en unidades posteriores.

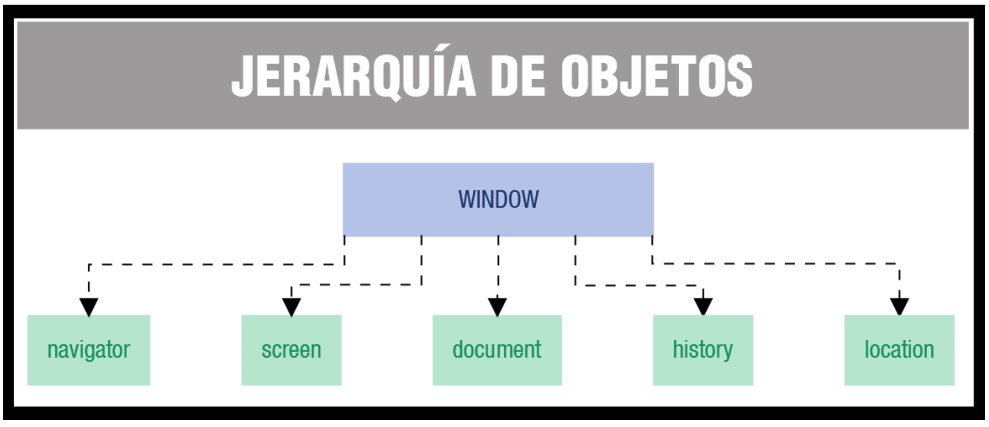
1.1. Objeto Windows

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

Este objeto 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 ve en el gráfico de la jerarquía de objetos, 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.



Atención: 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.

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

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

Como puedes ver, los parámetros van entre corchetes, indicando que son opcionales y que dependerán del método al que estemos llamando.

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] )
```

Podremos usar cualquiera de las dos referencias anteriores, pero intentaremos dejar la palabra reservada `self`, para scripts más complejos en los que tengamos múltiples marcos y ventanas.

Debido a que el objeto `window` siempre estará presente cuando ejecutemos nuestros scripts, podremos omitirlo, en referencias a los objetos dentro de esa ventana. Así que, si escribimos:

```
nombrePropiedad
nombreMétodo( [parámetros] )
```

También funcionaría sin ningún problema, porque se asume que esas propiedades o métodos, son del objeto de mayor jerarquía (el objeto `window`) en el cuál nos encontramos.

1.1.1. Gestion 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 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 si 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>
</html>
```

1.1.2. Propiedades y métodos

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 para cada marco.

Propiedades:

- **closed**. Válida a partir de Netscape 3 en adelante y MSIE 4 en adelante. Es un booleano que nos dice si la ventana está cerrada (closed = true) o no (closed = false).
- **defaultStatus**. Cadena que contiene el texto por defecto que aparece en la barra de estado (status bar) del navegador.
- **frames**. Es un array: cada elemento de este array (frames[0], frames[1], ...) es uno de los frames que contiene la ventana. Su orden se asigna según se definen en el documento HTML.
- **history**. Se trata de un array que representa las URLS visitadas por la ventana (están almacenadas en su historial).
- **length**. Variable que nos indica cuántos frames tiene la ventana actual.
- **location**. Cadena con la URL de la barra de dirección.
- **name**. Contiene el nombre de la ventana, o del frame actual.

- **opener**. Es una referencia al objeto window que lo abrió, si la ventana fue abierta usando el método open() que veremos cuando estudiemos los métodos.
- **parent**. Referencia al objeto window que contiene el frameset.
- **self**. Es un nombre alternativo del window actual.
- **status**. String con el mensaje que tiene la barra de estado.
- **top**. Nombre alternativo de la ventana del nivel superior.
- **window**. Igual que self: nombre alternativo del objeto window actual.

Métodos:

- **alert(mensaje)**. Muestra el mensaje 'mensaje' en un cuadro de diálogo
- **blur()**. Elimina el foco del objeto window actual. A partir de NS 3, IE 4.
- **clearInterval(id)**. Elimina el intervalo referenciado por 'id' (ver el método setInterval(), también del objeto window). A partir de NS 4, IE 4.
- **clearTimeout(nombre)**. Cancela el intervalo referenciado por 'nombre' (ver el método setTimeout(), también del objeto window).
- **close()**. Cierra el objeto window actual.
- **confirm(mensaje)**. Muestra un cuadro de diálogo con el mensaje 'mensaje' y dos botones, uno de aceptar y otro de cancelar. Devuelve true si se pulsa aceptar y devuelve false si se pulsa cancelar.
- **focus()**. Captura el foco del ratón sobre el objeto window actual. A partir de NS 3, IE 4.
- **moveBy(x,y)**. Mueve el objeto window actual el número de pixels especificados por (x,y). A partir de NS 4.
- **moveTo(x,y)**. Mueve el objeto window actual a las coordenadas (x,y). A partir de NS 4.
- **open(URL,nombre,caracteristicas)**. Abre la URL que le pasemos como primer parámetro en una ventana de nombre 'nombre'. Si esta ventana no existe, abrirá una ventana nueva en la que mostrará el contenido con las características especificadas. Las características que podemos elegir para la ventana que queramos abrir son las siguientes:
    - **toolbar** = [yes|no|1|0]. Nos dice si la ventana tendrá barra de herramientas (yes,1) o no la tendrá (no,0).
    - **location** = [yes|no|1|0]. Nos dice si la ventana tendrá campo de localización o no.
    - **directories** = [yes|no|1|0]. Nos dice si la nueva ventana tendrá botones de dirección o no.
    - **status** = [yes|no|1|0]. Nos dice si la nueva ventana tendrá barra de estado o no.
    - **menubar** = [yes|no|1|0]. Nos dice si la nueva ventana tendrá barra de menús o no.
    - **scrollbars** = [yes|no|1|0]. Nos dice si la nueva ventana tendrá barras de desplazamiento o no.
    - **resizable** = [yes|no|1|0]. Nos dice si la nueva ventana podrá ser cambiada de tamaño (con el ratón) o no.
    - **width** = px. Nos dice el ancho de la ventana en pixels.
    - **height** = px. Nos dice el alto de la ventana en pixels.
    - **outerWidth** = px. Nos dice el ancho ***total*** de la ventana en pixels. A partir de NS 4.
    - **outerHeight** = px. Nos dice el alto ***total*** de la ventana el pixels. A partir de NS 4
    - **left** = px. Nos dice la distancia en pixels desde el lado izquierdo de la pantalla a la que se debe colocar la ventana.
    - **top** = px. Nos dice la distancia en pixels desde el lado superior de la pantalla a la que se debe colocar la ventana.
- **prompt(mensaje,respuesta_por_defecto)**. Muestra un cuadro de diálogo que contiene una caja de texto en la cual podremos escribir una respuesta a lo que nos pregunte en '*mensaje*'. El parámetro '*respuesta_por_defecto*' es opcional, y mostrará la respuesta por defecto indicada al abrirse el cuadro de diálogo. El método retorna una cadena de caracteres con la respuesta introducida.
- **scroll(x,y)**. Desplaza el objeto window actual a las coordenadas especificadas por (x,y). A partir de NS3, IE4.
- **scrollBy(x,y)**. Desplaza el objeto window actual el número de pixels especificado por (x,y). A partir de NS4.

- **scrollTo(x,y)**. Desplaza el objeto window actual a las coordenadas especificadas por (x,y). A partir de NS4.
- **setInterval(expresion,tiempo)**. Evalua la expresión especificada después de que hayan pasado el número de milisegundos especificados en tiempo. Devuelve un valor que puede ser usado como identificativo por clearInterval(). A partir de NS4, IE4.
- **setTimeout(expresion,tiempo)**. Evalua la expresión especificada después de que hayan pasado el número de milisegundos especificados en tiempo. Devuelve un valor que puede ser usado como identificativo por clearTimeout(). A partir de NS4, IE4.

Ejemplo:

```
<HTML>
<HEAD>
<title>Ejemplo de JavaScript</title>
<script LANGUAGE="JavaScript">
<!—
var opciones="left=100,top=100,width=250,height=150", i= 0;
mi_ventana = window.open("","",opciones);
mi_ventana.document.write("Una prueba de abrir ventanas");
mi_ventana.moveTo(400,100);
moverVentana();


function moverVentana(){
        mi_ventana.moveBy(5,5);
        i++;
        if (i<20)
        setTimeout('moverVentana()',100);
        else
        mi_ventana.close();
}
//-->
</script>
</HEAD>
<BODY>
</BODY>
</HTML>
```

1.2. 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:

- **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, dentro de la URL.
- **hostname**. Cadena que contiene el nombre de dominio del servidor (o la dirección IP), dentro de la URL.
- **href**. Cadena que contiene la URL completa.
- **pathname**. Cadena que contiene el camino al recurso, dentro de la URL.
- **port**. Cadena que contiene el número de puerto del servidor, dentro de la URL.

- **protocol**. Cadena que contiene el protocolo utilizado (incluyendo los dos puntos), dentro de la URL.
- **search**. Cadena que contiene la información pasada en una llamada a un script, dentro de la URL.

Métodos:

- **reload()**. Vuelve a cargar la URL especificada en la propiedad href del objeto location.
- **replace(cadenaURL)**. Reemplaza el historial actual mientras carga la URL especificada en cadenaURL.

Ejemplo:

```
<HTML>
<HEAD>
<title>Ejemplo de JavaScript</title>
</HEAD>
<BODY>
<script LANGUAGE="JavaScript">
<!--
document.write("Location <b>href</b>: " + location.href + "<br>");
document.write("Location <b>host</b>: " + location.host + "<br>");
document.write("Location <b>hostname</b>: " + location.hostname + "<br>");
document.write("Location <b>pathname</b>: " + location.pathname + "<br>");
document.write("Location <b>port</b>: " + location.port + "<br>");
document.write("Location <b>protocol</b>: " + location.protocol + "<br>");
//-->
</script>
</BODY>
</HTML>
```

1.3. Objeto navigator

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

Propiedades:

- **appCodeName**. Cadena que contiene el nombre del código del cliente.
- **appName**. Cadena que contiene el nombre del cliente.
- **appVersion**. Cadena que contiene información sobre la versión del cliente.
- **language**. Cadena de dos caracteres que contiene información sobre el idioma de la versión del cliente.
- **mimeTypes**. Array que contiene todos los tipos MIME soportados por el cliente. A partir de NS 3.
- **platform**. Cadena con la plataforma sobre la que se está ejecutando el programa cliente.
- **plugins**. Array que contiene todos los plug-ins soportados por el cliente. A partir de NS 3.
- **userAgent**. Cadena que contiene la cabecera completa del agente enviada en una petición HTTP. Contiene la información de las propiedades appCodeName y appVersion.

Métodos:

- **javaEnabled()**. Devuelve true si el cliente permite la utilización de Java, en caso contrario, devuelve false.

Ejemplo:

```
<HTML>
<HEAD>
<title>Ejemplo de JavaScript</title>
<script LANGUAGE="JavaScript">
<!--
document.write("Navigator <b>appCodeName</b>: " + navigator.appCodeName + "<br>");
document.write("Navigator <b>appName</b>: " + navigator.appName + "<br>");
document.write("Navigator <b>appVersion</b>: " + navigator.appVersion + "<br>");
document.write("Navigator <b>language</b>: " + navigator.language + "<br>");
document.write("Navigator <b>platform</b>: " + navigator.platform + "<br>");
document.write("Navigator <b>userAgent</b>: " + navigator.userAgent + "<br>");
//-->
</script>
</HEAD>
<BODY>

</BODY>
</HTML>
```

1.4. Objeto documento

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 a la `window` actual).

Colecciones:

- **anchors[]** Es un array que contiene todos los hiperenlaces del documento.
- **applets[]** Es un array que contiene todos los applets del documento
- **forms[]** Es un array que contiene todos los formularios del documento.
- **images[]** Es un array que contiene todas las imágenes del documento.
- **links[]** Es un array que contiene todos los enlaces del documento.

Propiedades:

- **cookie** Devuelve todos los nombres/valores de las cookies en el documento.
- **domain** Cadena que contiene el nombre de dominio del servidor que cargó el documento.
- **lastModified** Devuelve la fecha y hora de la última modificación del documento
- **readyState** Devuelve el estado de carga del documento actual
- **referrer** Cadena que contiene la URL del documento desde el cuál llegamos al documento actual.
- **title** Devuelve o ajusta el título del documento.
- **URL** Devuelve la URL completa del documento.

Métodos:

- **close()** Cierra el flujo abierto previamente con document.open().
- **getElementById()** Para acceder a un elemento identificado por el id escrito entre paréntesis.

- **getElementsByName()** Para acceder a los elementos identificados por el atributo name escrito entre paréntesis.
- **getElementsByTagName()** Para acceder a los elementos identificados por el tag o la etiqueta escrita entre paréntesis.
- **open()** Abre el flujo de escritura para poder utilizar document.write() o document.writeln en el documento.
- **write()** Para poder escribir expresiones HTML o código de JavaScript dentro de un documento.
- **writeln()** Lo mismo que write() pero añade un salto de línea al final de cada instrucción.

2. Marcos

Caso práctico

Después de estudiar los objetos del apartado anterior **Antonio** ha decidido aplicarse en utilizar los marcos e **iframes** aunque ya sabe que no se implementan mucho es siempre interesante comprender porque han dejado de utilizarse e incluso puede decirse que están en desuso, pero por si algún día le toca mantener algún sitio antiguo y porqué el saber no ocupa lugar. Ha estado pensando que podría abrir varias veces la página de YouTube.com y mostrarla en la misma página web. Para poder hacerlo deberá estudiar las propiedades y métodos de los objetos implicados.

De esta forma podrá gestionar los distintos marcos y poder realizar comunicaciones con los mismos.

Diréis que esto es una perdida de tiempo, que como ya sabe el alumno Antonio está desfasado y no sirve. Para nada. Si sois capaces de comprender la forma de comunicación entre marcos os será mucho más fácil comprender el tema que todos estáis deseando aprender: AJAX. Su necesidad, su utilidad. Y aún así, si queréis introducir otra página web en la vuestra, es la forma más sencilla.
Por ejemplo, podéis tener un botón que cuando se pulse cambie la página a la que está indicada por ese botón. Es muy satisfactorio, aunque no utilicéis páginas de google, que la mayoría no funcionan en `iframes`.
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>` en un documento HTML, se creará un objeto `frame`.
Todo lo anterior se aplicará también al objeto `Iframe` `<iframe>`.

Propiedades:

- **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 marginwidth (ancho del margen) de un frame/iframe.
- **marginWidth**:
- **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.
- **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.

Métodos:

- **Onload**: Script que se ejecutará inmediatamente después a que se cargue el frame/iframe.

2.1. Jerarquias

Uno de los aspectos más atractivos de JavaScript en aplicaciones cliente, es que permite interacciones del usuario en 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.

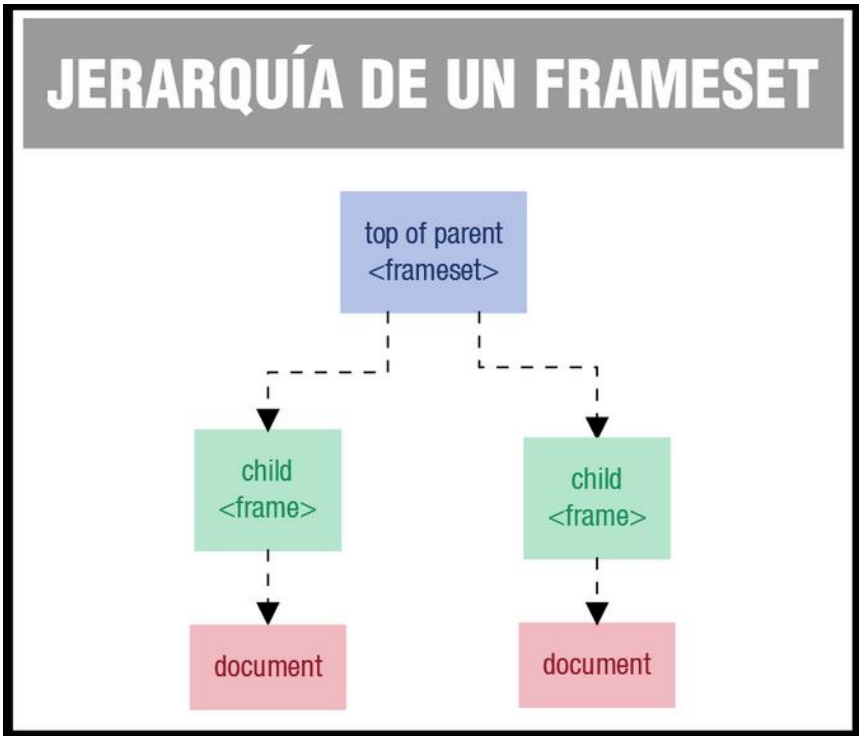
En el gráfico de jerarquías de objetos, viste como el objeto `window` está en la cabeza de la jerarquía y puede tener sinónimos como `self`. En esta sección veremos que, cuando trabajamos con marcos o iframes, podemos referenciar a las ventanas como: `frame`, `top` y `parent`.

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. Ésto 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 no tenemos referencia de la sección del portal o web en la que nos encontramos.

Ejemplo:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
"http://www.w3.org/TR/html4/frameset.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Titulo para todas las páginas de este conjunto de Frames</title>
</head>
<frameset cols="50%,50%">
  <frame name="frameIzdo" src="documento1.html" title="Frame 1">
  <frame name="frameDcho" src="documento2.html" title="Frame 2">
<noframes></noframes>
</frameset>
</html>
```

Este código divide la ventana del navegador en dos marcos de igual tamaño, con dos documentos diferentes en cada columna. Un 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). Véase la siguiente figura de la jerarquía resultante:



Fíjate en el gráfico 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.) y son los frames hijos, los que sí tienen objeto `document`. El objeto `document` de un marco, es independiente del objeto `document` del otro marco, y en realidad cada uno de los marcos, será un objeto `window` independiente.

Veamos `iframe`. Los iframes pueden encontrarse dentro del `body` y son por lo tanto un poco más flexibles. Veamos un ejemplo:

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="UTF-8">
   <title>Frames e iFrames</title>
</head>
<body>
  <h1>iFrames</h1>
  <iframe width="45%" height="400px"
src="http://terra.es" name="iframe_terra"></iframe>
  <iframe width="45%" height="400px"
src="http://firerfox.com" name="iframe_fire"></iframe>
</body>
</html>
```

2.2. Comunicación entre marcos

Referencias Padre-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. Podemos acceder a un marco a través de la sintaxis de array o

por el nombre que le hemos dado a ese marco, por el id o con el atributo `name` que hemos puesto en la marca `<frame>`.
Ejemplos de referencias a los marcos hijo:
(Recordar que todo lo que va entre corchetes [] es opcional).

```
[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 fácil.

Referencias Hijo-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:

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

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();
```

Además como la ventana padre está en el top de la jerarquía de ventanas, opcionalmente podríamos escribir:

```
var valor=top.NombreFuncion();
```

Referencias Hijos-Hijos.

El navegador necesita un poco más de asistencia cuando queremos que una ventana hija se comunique con una hermana. Una de las propiedades de cualquier ventana o marco es su padre. 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).

Podemos utilizar alguno de los siguientes formatos:

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

2.3. Comunicación entre múltiples ventanas

En esta sección, vamos a ver cómo podemos comunicarnos con sub-ventanas, que abrimos 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 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.
Ejemplo:

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="content-type" content="text/html;charset=utf-8">
<title>Ventana Principal</title>
<script type="text/javascript">
function abrirSubVentana()
{
	nuevaVentana = window.open("ejemplo2_1.html", "sub", "height=300,width=400");
}

function cerrarSubVentana()
{
	if (nuevaVentana)
	{
		nuevaVentana.close();
	}
}
</script>
</head>
<body>
<h1>Ventana padre - principal</h1>
<form action="">
<p>
<input type="button" value="Abrir sub ventana" id="abrir" onclick="abrirSubVentana()">
<input type="button" value="Cerrar sub ventana" id="cerrar" onclick="cerrarSubVentana()">
</p>
<p>
<label>Texto proveniente de la sub-ventana:</label>
<input type="text" id="original">
</p>
</form>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="content-type" content="text/html;charset=utf-8">
<title>Sub-Documento</title>
<script type="text/javascript">
```

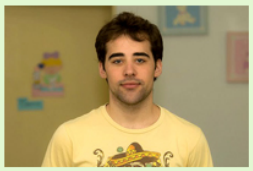
```
function  copiarAlPadre()
{
        opener.document.getElementById("original").value=
document.getElementById('textocopiar').value;
}
</script>
</head>
<body>
<h1>Sub-Ventana</h1>
<form  id="formulario">
<p>
<label  for="textocopiar">Introduzca texto a copiar en la ventana principal:</label>
<input  type="text"  id="textocopiar"/>
</p>
<p>
<input  type="button"  value="Enviar texto a la ventana padre"  id="enviar" onclick="copiarAlPadre()"
/>
</p>
</form>
</body>
</html>
```

Si no se abren las ventanas del ejemplo anterior, a lo mejor tienes que desactivar el bloqueador de pop-ups y volver a probar.

3. Objetos nativos en javascript

Caso práctico

El lenguaje JavaScript es un lenguaje basado en objetos. A **Antonio** le suena un poco el tema de objetos aunque nunca trabajó intensivamente con ellos. Como todos los lenguajes que incorporan sus funciones para realizar acciones, conversiones, etc., en JavaScript también dispone de unos objetos nativos que le van a permitir a **Antonio** el realizar muchas de esas tareas.



Estos objetos, hacen referencia al trabajo con cadenas de texto, operaciones matemáticas, números, valores booleanos y trabajo con fechas y horas.

Ésto le va a ser muy útil para realizar su aplicación ya que tendrá que realizar diferentes tipos de conversiones de datos, trabajar intensivamente con cadenas y por supuesto con fechas y horas.

Aunque no hemos visto como crear objetos, sí que ya hemos dado unas pinceladas a lo que son los objetos, propiedades y métodos.
En esta sección vamos a echar una ojeada a objetos que son nativos en JavaScript, ésto es, aquello que JavaScript nos da, listos para su utilización en nuestra aplicación.
Echaremos un vistazo a los objetos `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 cuales 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 las comillas dobles o simples:

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

Si queremos emplear comillas dobles al principio y fin de la cadena, y que en el contenido aparezcan también comillas dobles, tendríamos que escaparlas con \", por ejemplo:

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

Cuando estamos hablando de cadenas muy largas, podríamos concatenarlas con + =, por ejemplo:

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

Si queremos concatenar el contenido de una variable dentro de una cadena de texto emplearemos el símbolo + :

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

Caracteres especiales o caracteres de escape.
La forma en la que se crean las cadenas en JavaScript, hace que cuando tengamos que emplear ciertos caracteres especiales en una cadena de texto, tengamos que escaparlos, empleando el símbolo \ seguido del carácter.
Vemos aquí un listado de los caracteres especiales o de escape en JavaScript:

3.1.1. Propiedades y métodos

Propiedades:

- **constructor** Devuelve la función que ha creado el prototipo del objeto String
- **length** Devuelve la longitud de una cadena
- **prototype** Te permite añadir propiedades y métodos a un objeto

Métodos:

- **charAt()** Devuelve el carácter en el índice especificado
- **charCodeAt()** Devuelve el carácter Unicode del índice especificado
- **concat()** Une dos o más cadenas y devuelve una copia de las cadenas unidas
- **fromCharCode()** Convierte valores Unicode a caracteres
- **indexOf()** Devuelve la posición de la primera aparición de un valor especificado en una cadena
- **lastIndexOf()** Devuelve la posición de la última aparición de un valor especificado en una cadena

- **match()** Busca una coincidencia entre una expresión regular y una cadena, y devuelve las coincidencias
- **replace()** Busca una coincidencia entre una subcadena (o expresión regular) y una cadena, y sustituye a la subcadena encontrada con una nueva subcadena
- **search()** Busca una coincidencia entre una expresión regular y una cadena, y devuelve la posición de la coincidencia
- **slice()** Extrae una parte de una cadena y devuelve una nueva cadena
- **split()** Divide una cadena dentro de un array de subcadenas
- **substr()** Extrae los caracteres de una cadena, empezando en la posición de inicio especificado, y el número especificado de caracteres
- **substring()** Extrae los caracteres de una cadena, entre dos indices especificados
- **toLowerCase()** Convierte una cadena a minúsculas
- **toUpperCase()** Convierte una cadena a mayúsculas
- **valueOf()** Devuelve el valor primitive de un objeto String

3.2. Objeto Math

Ya vimos anteriormente algunas funciones, que nos permitían convertir cadenas a diferentes formatos numéricos (`parseInt`, `parseFloat`). A parte 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:

- **E** Devuelve el número Euler (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 base 2 de E (aproximadamente 1.442).
- **LOG10E** Devuelve el logaritmo base 10 de E (aproximadamente 0.434).
- **PI** Devuelve el número PI (aproximadamente 3.14159).
- **SQRT2** Devuelve la raíz cuadrada de 2 (aproximadamente 1.414).

Métodos:

- **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 alta hacia el siguiente entero.
- **cos(x)** Devuelve el coseno de x (x está en radianes).
- **floor(x)** Devuelve el número x redondeado a la baja hacia el anterior entero.
- **log(x)** Devuelve el logaritmo neperiando (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.

Ejemplos:

```
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 />");
```

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.
Lo primero, es que el objeto `Number` contiene propiedades que nos indican el rango de números soportados en el lenguaje. El número más alto es 1.79E + 308; el número más bajo es 2.22E-308. 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:

- **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).
- **POSITIVE_INFINITY** Representa a infinito positivo (se devuelve en caso de overflow).
- **prototype** Permite añadir nuestras propias propiedades y métodos a un objeto.

Métodos:

- **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 la longitud x.
- **toString()** Convierte un objeto **Number** en una cadena.
    - Si se pone 2 como parámetro se mostrará el número en binario.
    - Si se pone 8 como parámetro se mostrará el número en octal.
    - Si se pone 16 como parámetro se mostrará el número en
- **valueOf()** Devuelve el valor primitivo de un objeto Number.

Ejemplo:

```
var num = new Number(13.3714);
document.write(num.toPrecision(3)+"<br />");
document.write(num.toFixed(1)+"<br />");
document.write(num.toString(2)+"<br />");
document.write(num.toString(8)+"<br />");
document.write(num.toString(16)+"<br />");
document.write(Number.MIN_VALUE);
document.write(Number.MAX_VALUE);
```

3.4. Objeto Boolean

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

Propiedades:

- **constructor** Devuelve la función que creó el objeto Boolean.
- **prototype** Te permitirá añadir propiedades y métodos a un objeto

Métodos:

- **toString()** Convierte un valor Boolean a una cadena y devuelve el resultado.
- **valueOf()** Devuelve el valor primitivo de un objeto Boolean.

Ejemplos:

```
var bool = new Boolean(1);
document.write(bool.toString());
document.write(bool.valueOf());
```

La clase `Boolean` es una clase nativa de JavaScript que nos permite crear valores booleanos.
Una de sus posibles utilidades es la de conseguir valores boleanos a partir de datos de cualquier otro tipo. No obstante, al igual que ocurría con la clase `Number`, es muy probable que no la llegues a utilizar nunca.
Dependiendo de lo que reciba el constructor de la clase `Boolean` el valor del objeto boleano que se crea será verdadero o falso, de la siguiente manera:
- **Se inicializa a false** cuando no pasas ningún valor al constructor, o si pasas una cadena vacía, el número 0 o la palabra `false` sin comillas.
- **Se inicializa a true** cuando recibe cualquier valor entrecomillado o cualquier número distinto de 0.

Se puede comprender el funcionamiento de este objeto fácilmente si examinamos unos ejemplos.

```
var b1 = new Boolean()
document.write(b1 + "<br>")
//muestra false
var b2 = new Boolean("")
document.write(b2 + "<br>")
//muestra false
var b25 = new Boolean(false)
document.write(b25 + "<br>")
//muestra false
var b3 = new Boolean(0)
document.write(b3 + "<br>")
//muestra false
var b35 = new Boolean("0")
```

```
document.write(b35 + "<br>")
//muestra true
var b4 = new Boolean(3)
document.write(b4 + "<br>")
//muestra true
var b5 = new Boolean("Hola")
document.write(b5 + "<br>")
//muestra true
```

3.5. Objeto Date

El objeto `Date` se utiliza para trabajar con fechas y horas. Los objetos `Date` se crean con `new Date()`. Hay 4 formas de instanciar (crear un objeto de tipo `Date`):

```
var d = new Date();
var d = new Date(milisegundos);
var d = new Date(cadena de Fecha);
var d = new Date(año, mes, día, horas, minutos, segundos, milisegundos);
// (el mes comienza en 0, Enero sería 0, Febrero 1, etc.)
```

Propiedades:

- **constructor** Devuelve la función que creó el objeto Date.
- **prototype** Te permitirá añadir propiedades y métodos a un objeto.

Métodos:

- **getDate()** Devuelve el día del mes (de 1-31).
- **getDay()** Devuelve el día de la semana (de 0-6).
- **getFullYear()** Devuelve el año (4 dígitos).
- **getHours()** Devuelve la hora (de 0-23).
- **getMilliseconds()** Devuelve los milisegundos (de 0-999).
- **getMinutes()** Devuelve los minutos (de 0-59).
- **getMonth()** Devuelve el mes (de 0-11).
- **getSeconds()** Devuelve los segundos (de 0-59).
- **getTime()** Devuelve 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()** Devuelve 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()** Devuelve el año en base a la hora UTC (4 dígitos).
- **setDate()** Ajusta el día del mes del objeto (de 1-31).
- **setFullYear()** Ajusta el año del objeto (4 dígitos).
- **setHours()** Ajusta la hora del objeto (de 0-23).

Ejemplo:

```
var d = new Date();
document.write(d.getFullYear());
document.write(d.getMonth());
document.write(d.getUTCDay());
var d2 = new Date(5,28,2011,22,58,00);
d2.setMonth(0);
d.setFullYear(2020);
```