

Modelo de objetos predefinidos en JavaScript.

Caso práctico

Antonio ha completado correctamente la fase de introducción y fundamentos básicos del lenguaje JavaScript, y ahora comienza a investigar en las características de los objetos predefinidos en JavaScript.

Estos objetos le van a permitir gestionar ventanas, marcos, propiedades de los navegadores, de las URL, etc. en JavaScript.

Además, también va a poder realizar operaciones matemáticas, de fecha y de cadenas, con otros tantos objetos nativos del lenguaje JavaScript.

Antonio tiene una pequeña reunión con **Ada** y con su tutor **Juan**, para comentar los progresos realizados hasta este momento y se pone manos a la obra con esta nueva sección.



Materiales formativos de FP Online propiedad del Ministerio de Educación, Cultura y Deporte.

1.- Objetos de más alto nivel en Javascript.

Caso práctico

Bajo la tutoría de **Juan, Antonio** se dispone a profundizar en los objetos básicos y de más alto nivel de JavaScript. Estos objetos, los encontrará en prácticamente la mayoría de aplicaciones que haga con JavaScript, por lo que será fundamental, que tenga muy claras las características y diferentes funcionalidades que estos objetos le van a aportar a sus aplicaciones.



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 contruidos 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:

```
nombreaketo.propiedad
```

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

```
nombreaketo.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.

En esta unidad, verás solamente las propiedades básicas, y los métodos de los objetos mencionados anteriormente.

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



Es muy importante que tengas este gráfico en mente porque va a ser la guía a lo largo de toda esta unidad.

Autoevaluación

El nombre de un método en JavaScript siempre lleva paréntesis al final:

- ☐ Sí.
- ☐ No.
- ☐ Depende de si lleva o no parámetros.

Es correcta. Todos los métodos de cualquier objeto en JavaScript se nombrarán con el nombre del método seguido de paréntesis y sus parámetros opcionales.

Es incorrecta. Si no lleva paréntesis se consideraría una propiedad.

No es correcta. Aunque no lleve parámetros tendría que llevar los paréntesis igualmente.

Solución

1. Opción correcta
2. Incorrecto
3. Incorrecto

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 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.nombreMetodo( [parametros] )
```

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.nombreMetodo( [parametros] )
```

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  
nombreMetodo( [parametros] )
```

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.

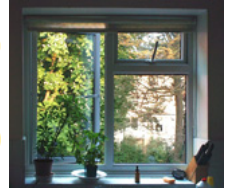
Citas para pensar

“Sólo cerrando las puertas detrás de uno se abren ventanas hacia el porvenir.”

SAGAN, Françoise.

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 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>
```

[Descarga del código del ejemplo.](#) (0.01 MB)

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 cada marco.



Propiedades del objeto Window

Propiedad	Descripción
<code>closed</code>	Devuelve un valor Boolean indicando cuando una ventana ha sido cerrada o no.
<code>defaultStatus</code>	Ajusta o devuelve el valor por defecto de la barra de estado de una ventana.
<code>document</code>	Devuelve el objeto <code>document</code> para la ventana.
<code>frames</code>	Devuelve un array de todos los marcos (incluidos <code>iframes</code>) de la ventana actual.
<code>history</code>	Devuelve el objeto <code>history</code> de la ventana.
<code>length</code>	Devuelve el número de frames (incluyendo <code>iframes</code>) que hay en dentro de una ventana.
<code>location</code>	Devuelve la Localización del objeto ventana (URL del fichero).
<code>name</code>	Ajusta o devuelve el nombre de una ventana.
<code>navigator</code>	Devuelve el objeto <code>navigator</code> de una ventana.
<code>opener</code>	Devuelve la referencia a la ventana que abrió la ventana actual.
<code>parent</code>	Devuelve la ventana padre de la ventana actual.
<code>self</code>	Devuelve la ventana actual.
<code>status</code>	Ajusta el texto de la barra de estado de una ventana.

Métodos del objeto Window

Método	Descripción
<code>alert()</code>	Muestra una ventana emergente de alerta y un botón de aceptar.
<code>blur()</code>	Elimina el foco de la ventana actual.
<code>clearInterval()</code>	Resetea el cronómetro ajustado con <code>setInterval()</code> .
<code>setInterval()</code>	Llama a una función o evalúa una expresión en un intervalo especificado (en milisegundos).
<code>close()</code>	Cierra la ventana actual.
<code>confirm()</code>	Muestra una ventana emergente con un mensaje, un botón de aceptar y un botón de cancelar.
<code>focus()</code>	Coloca el foco en la ventana actual.
<code>open()</code>	Abre una nueva ventana de navegación.
<code>prompt()</code>	Muestra una ventana de diálogo para introducir datos.

Para saber más

El siguiente enlace amplía información sobre el objeto `window` y todas sus propiedades y métodos.

[Más información y ejemplo sobre el objeto window](#)

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`.



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
<code>hash</code>	Cadena que contiene el nombre del enlace, dentro de la URL.
<code>host</code>	Cadena que contiene el nombre del servidor y el número del puerto, dentro de la URL.
<code>hostname</code>	Cadena que contiene el nombre de dominio del servidor (o la dirección IP), dentro de la URL.
<code>href</code>	Cadena que contiene la URL completa.
<code>pathname</code>	Cadena que contiene el camino al recurso, dentro de la URL.
<code>port</code>	Cadena que contiene el número de puerto del servidor, dentro de la URL.
<code>protocol</code>	Cadena que contiene el protocolo utilizado (incluyendo los dos puntos), dentro de la URL.
<code>search</code>	Cadena que contiene la información pasada en una llamada a un script, dentro de la URL.

Métodos del objeto Location

<code>assign()</code>	Carga un nuevo documento.
<code>reload()</code>	Vuelve a cargar la URL especificada en la propiedad <code>href</code> del objeto <code>location</code> .
<code>replace()</code>	Reemplaza el historial actual mientras carga la URL especificada en <code>cadenaURL</code> .

Citas para pensar

“Mil rutas se apartan del fin elegido, pero hay una que llega a él.”

MONTAIGNE, Michel de.

Para saber más

El siguiente enlace amplía información sobre el objeto `Location` y todas sus propiedades y métodos.

[Más información y ejemplos sobre el objeto location](#)

1.3.- Objeto navigator.

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



Propiedades del objeto Navigator

Propiedad	Descripción
<code>appName</code>	Cadena que contiene el nombre en código del navegador.
<code>appVersion</code>	Cadena que contiene el nombre del cliente.
<code>cookieEnabled</code>	Cadena que contiene información sobre la versión del cliente.
<code>platform</code>	Determina si las cookies están o no habilitadas en el navegador.
<code>userAgent</code>	Cadena con la plataforma sobre la que se está ejecutando el programa cliente.
<code>userAgent</code>	Cadena que contiene la cabecera completa del agente enviada en una petición HTTP. Contiene la información de las propiedades <code>appName</code> y <code>appVersion</code> .

Métodos del objeto Navigator

Método	Descripción
<code>javaEnabled()</code>	Devuelve true si el cliente permite la utilización de Java, en caso contrario, devuelve false.

Para saber más

El siguiente enlace amplía información sobre el objeto `Navigator` y todas sus propiedades y métodos.

[Más información y ejemplos sobre el objeto navigator.](#)

Autoevaluación

Si queremos introducir datos a través de una ventana de diálogo en nuestra aplicación de JavaScript lo haremos con:

- ☐ La propiedad `input` del objeto `window`.
- ☐ La propiedad `userAgent` del objeto `navigator`.
- ☐ El método `prompt` del objeto `window`.

No es correcta. No existe tal propiedad en el objeto `window`.

Incorrecta. Esa propiedad nos permite consultar información sobre el navegador utilizado.

Muy bien. Este método te permitirá solicitar datos al usuario, mediante una ventana de diálogo que podrás usar en tu aplicación.

Solución

1. Incorrecto
2. Incorrecto
3. Opción correcta

1.4.- 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 a la `window` actual).



Colecciones del objeto Document

Colección	Descripción
<code>anchors[]</code>	Es un array que contiene todos los hiperenlaces del documento.
<code>forms[]</code>	Es un array que contiene todos los formularios del documento.
<code>images[]</code>	Es un array que contiene todas las imágenes del documento.
<code>links[]</code>	Es un array que contiene todos los enlaces del documento.

Propiedades del objeto Document

Propiedad	Descripción
<code>cookie</code>	Devuelve todos los nombres/valores de las cookies en el documento.
<code>domain</code>	Cadena que contiene el nombre de dominio del servidor que cargó el documento.
<code>referrer</code>	Cadena que contiene la URL del documento desde el cuál llegamos al documento actual.
<code>title</code>	Devuelve o ajusta el título del documento.
<code>URL</code>	Devuelve la URL completa del documento.

Propiedades del objeto Document

Método	Descripción
<code>close()</code>	Cierra el flujo abierto previamente con <code>document.open()</code> .
<code>getElementById()</code>	Para acceder a un elemento identificado por el <code>id</code> escrito entre paréntesis.
<code>getElementsByName()</code>	Para acceder a los elementos identificados por el atributo <code>name</code> escrito entre paréntesis.
<code>getElementsByTagName()</code>	Para acceder a los elementos identificados por el <code>tag</code> o la etiqueta escrita entre paréntesis.
<code>open()</code>	Abre el flujo de escritura para poder utilizar <code>document.write()</code> O <code>document.writeln</code> en el documento.
<code>write()</code>	Para poder escribir expresiones HTML o código de JavaScript dentro de un documento.
<code>writeln()</code>	Lo mismo que <code>write()</code> pero añade un salto de línea al final de cada instrucción.

Para saber más

El siguiente enlace amplía información sobre el objeto `Document` todas sus propiedades y métodos.

[Más información y ejemplos sobre el objeto Document.](#)

2.- Marcos.

Caso práctico

Antonio ha estado estudiando los métodos y propiedades de varios objetos, y ha llegado el momento de investigar un poco más en los marcos y los iframes, que aunque están cada vez más en desuso, pueden resultar interesantes para realizar algunas tareas del proyecto, por ejemplo las que impliquen el mostrar varias páginas simultáneamente en una misma ventana.

Verá los objetos, con sus propiedades y métodos, que le van a permitir gestionar diferentes marcos y poder realizar una comunicación entre ellos.



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 del objeto Frame/Iframe

Propiedad	Descripción
<code>align</code>	Cadena que contiene el valor del atributo <code>align</code> (alineación) en un <code>iframe</code> .
<code>contentDocument</code>	Devuelve el objeto documento contenido en un <code>frame/iframe</code> .
<code>contentWindow</code>	Devuelve el objeto <code>window</code> generado por un <code>frame/iframe</code> .
<code>frameBorder</code>	Cadena que contiene el valor del atributo <code>frameborder</code> (borde del marco) de un <code>frame/iframe</code> .
<code>height</code>	Cadena que contiene el valor del atributo <code>height</code> (altura) de un <code>iframe</code> .
<code>longDesc</code>	Cadena que contiene el valor del atributo <code>longdesc</code> (descripción larga) de un <code>frame/iframe</code> .
<code>marginHeight</code>	Cadena que contiene el valor del atributo <code>marginheight</code> (alto del margen) de un <code>frame/iframe</code> .
<code>marginWidth</code>	Cadena que contiene el valor del atributo <code>marginwidth</code> (ancho del margen) de un <code>frame/iframe</code> .
<code>name</code>	Cadena que contiene el valor del atributo <code>name</code> (nombre) de un <code>frame/iframe</code> .
<code>noResize</code>	Cadena que contiene el valor del atributo <code>noresize</code> de un <code>frame/iframe</code> .
<code>scrolling</code>	Cadena que contiene el valor del atributo <code>scrolling</code> (desplazamiento) de un <code>frame/iframe</code> .
<code>src</code>	Cadena que contiene el valor del atributo <code>src</code> (origen) de un <code>frame/iframe</code> .
<code>width</code>	Cadena que contiene el valor del atributo <code>width</code> (ancho) de un <code>iframe</code> .

Eventos del objeto Frame/Iframe

Evento	Descripción
<code>onload</code>	Script que se ejecutará inmediatamente después a que se cargue el <code>frame/iframe</code> .

Citas para pensar

“Los objetos son los amigos que ni el tiempo, ni la belleza, ni la fidelidad consiguen alterar.”

SAGAN, Françoise.

2.1.- Jerarquías.

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.

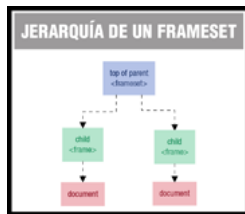
En el gráfico de jerarquías de objetos, viste como el objeto `window` está en la cabeza de la jerarquía y puede tener sinónimos como `self`. En esta sección veremos que, cuando trabajamos con marcos o iframes, podemos referenciar a las ventanas como: `frame`, `top` y `parent`.

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

Ejemplo de Frame:

```
<!DOCTYPE html>
<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:



Fijate 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.

Citas para pensar

“Una sociedad sin jerarquía es una casa sin escalera.”

DAUDET, Alphonse.

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 (`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).

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.

[Descarga de un ejemplo JavaScript con dos ventanas.](#) (0.01 MB)

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

The Ruffle emulator does not yet support
ActionScript 3, required by this content.
If you choose to run it anyway, interactivity will be
missing or limited.

More info

Run anyway

[Resumen textual alternativo](#)

Citas para pensar

“No pensábamos en el negocio, sino en Internet como una forma de comunicación global.”

YANG, Jerry.

Autoevaluación

Si queremos comunicar dos marcos que están en una misma ventana lo haremos:

- ☐ A través de su padre (`parent`).
- ☐ Directamente con el nombre del marco.
- ☐ A través del objeto `navigator`.

Correcto. A través de `parent` podremos conectar dos marcos hijos para poder acceder a las diferentes propiedades u objetos de cada uno de ellos.

No es correcta, porque dos marcos que son hermanos no conocen sus nombres directamente, si no es a través de la ventana padre.

Incorrecta. El objeto `navigator` sólo nos permite acceder a las propiedades del navegador, no a las ventanas o marcos que esté mostrando.

Solución

1. Opción correcta
2. Incorrecto
3. Incorrecto

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`.



Citas para pensar

“Si me hubieran hecho objeto sería objetivo, pero me hicieron sujeto.”

BERGAMÍN, José.

Reflexiona

¿Te has parado a pensar alguna vez que nuestro mundo está rodeado de objetos por todas partes?

¿Sabes que prácticamente, todos esos objetos tienen algunas propiedades como pueden ser tamaño, color, peso, tipo de corriente que usan, temperatura, tipo de combustible, etc.?

¿Sabes que también podemos realizar acciones con esos objetos, como pueden ser encender, apagar, mover, abrir, cerrar, subir temperatura, bajar temperatura, marcar número, colgar, etc.?

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:

Caracteres de escape y especiales en JavaScript

Símbolos	Explicación
<code>\"</code>	Comillas dobles.
<code>'</code>	Comilla simple.
<code>\</code>	Barra inclinada.

Símbolos	Explicación
	Retroceso.
	Tabulador.
	Nueva línea.
	Salto de línea.
	Avance de página.

3.1.1.- Propiedades y métodos del objeto String.



Para crear un objeto `String` lo podremos hacer de la siguiente forma:

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

O también se podría hacer:

```
var miCadena = "texto de la cadena";
```

Es decir, cada vez que tengamos una cadena de texto, en realidad es un objeto `String` que tiene propiedades y métodos:

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

Propiedades del objeto String

Propiedad	Descripción
<code>length</code>	Contiene la longitud de una cadena.

Métodos del objeto String

Métodos	Descripción
<code>charAt()</code>	Devuelve el carácter especificado por la posición que se indica entre paréntesis.
<code>charCodeAt()</code>	Devuelve <u>Unicode</u> del carácter especificado por la posición que se indica entre paréntesis.
<code>concat()</code>	Une una o más cadenas y devuelve el resultado de esa unión.
<code>fromCharCode()</code>	Convierte valores Unicode a caracteres.
<code>indexOf()</code>	Devuelve la posición de la primera ocurrencia del carácter buscado en la cadena.
<code>lastIndexOf()</code>	Devuelve la posición de la última ocurrencia del carácter buscado en la cadena.
<code>match()</code>	Busca una coincidencia entre una expresión regular y una cadena y devuelve las coincidencias o null si no ha encontrado nada.
<code>replace()</code>	Busca una subcadena en la cadena y la reemplaza por la nueva cadena especificada.
<code>search()</code>	Busca una subcadena en la cadena y devuelve la posición dónde se encontró.
<code>slice()</code>	Extrae una parte de la cadena y devuelve una nueva cadena.
<code>split()</code>	Divide una cadena en un array de subcadenas.
<code>substr()</code>	Extrae los caracteres de una cadena, comenzando en una determinada posición y con el número de caracteres indicado.
<code>substring()</code>	Extrae los caracteres de una cadena entre dos índices especificados.
<code>toLowerCase()</code>	Convierte una cadena en minúsculas.
<code>toUpperCase()</code>	Convierte una cadena en mayúsculas.

Ejemplos de uso:

```
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/>");
```

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 del objeto Math

Propiedad	Descripción
<code>E</code>	Devuelve el número Euler (aproximadamente 2.718).
<code>LN2</code>	Devuelve el logaritmo neperiano de 2 (aproximadamente 0.693).
<code>LN10</code>	Devuelve el logaritmo neperiano de 10 (aproximadamente 2.302).
<code>LOG2E</code>	Devuelve el logaritmo base 2 de E (aproximadamente 1.442).
<code>LOG10E</code>	Devuelve el logaritmo base 10 de E (aproximadamente 0.434).
<code>PI</code>	Devuelve el número PI (aproximadamente 3.14159).
<code>SQRT2</code>	Devuelve la raíz cuadrada de 2 (aproximadamente 1.414).

Métodos del objeto Math

Método	Descripción
<code>abs(x)</code>	Devuelve el valor absoluto de x.
<code>acos(x)</code>	Devuelve el arcocoseno de x, en radianes.
<code>asin(x)</code>	Devuelve el arcoseno de x, en radianes.
<code>atan(x)</code>	Devuelve el arcotangente de x, en radianes con un valor entre $-\pi/2$ y $\pi/2$.
<code>atan2(y,x)</code>	Devuelve el arcotangente del cociente de sus argumentos.
<code>ceil(x)</code>	Devuelve el número x redondeado al alta hacia el siguiente entero.
<code>cos(x)</code>	Devuelve el coseno de x (x está en radianes).
<code>floor(x)</code>	Devuelve el número x redondeado a la baja hacia el anterior entero.
<code>log(x)</code>	Devuelve el logaritmo neperiano (base E) de x.
<code>max(x,y,z,...,n)</code>	Devuelve el número más alto de los que se pasan como parámetros.
<code>min(x,y,z,...,n)</code>	Devuelve el número más bajo de los que se pasan como parámetros.
<code>pow(x,y)</code>	Devuelve el resultado de x elevado a y.
<code>random()</code>	Devuelve un número al azar entre 0 y 1.
<code>round(x)</code>	Redondea x al entero más próximo.
<code>sin(x)</code>	Devuelve el seno de x (x está en radianes).
<code>sqrt(x)</code>	Devuelve la raíz cuadrada de x.

Método	Descripción
<code>tan(x)</code>	Devuelve la tangente de un ángulo.

Ejemplos de uso:

```
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 del objeto Number

Propiedad	Descripción
<code>constructor</code>	Devuelve la función que creó el objeto <code>Number</code> .
<code>MAX_VALUE</code>	Devuelve el número más alto disponible en JavaScript.
<code>MIN_VALUE</code>	Devuelve el número más pequeño disponible en JavaScript.
<code>NEGATIVE_INFINITY</code>	Representa a infinito negativo (se devuelve en caso de overflow).
<code>POSITIVE_INFINITY</code>	Representa a infinito positivo (se devuelve en caso de overflow).
<code>prototype</code>	Permite añadir nuestras propias propiedades y métodos a un objeto.

Métodos del objeto Number

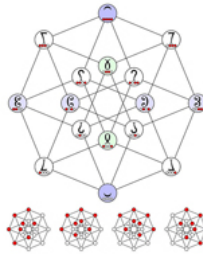
<code>toExponential(x)</code>	Convierte un número a su notación exponencial.
<code>toFixed(x)</code>	Formatea un número con x dígitos decimales después del punto decimal.
<code>toPrecision(x)</code>	Formatea un número a la longitud x.
<code>toString()</code>	Convierte un objeto <code>Number</code> en 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 hexadecimal.
<code>valueOf()</code>	Devuelve el valor primitivo de un objeto <code>Number</code> .

Algunos ejemplos de uso:

```
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 del objeto Boolean

constructor	Devuelve la función que creó el objeto <code>Boolean</code> .
prototype	Te permitirá añadir propiedades y métodos a un objeto.

Métodos del objeto Boolean

toString()	Convierte un valor <code>Boolean</code> a una cadena y devuelve el resultado.
valueOf()	Devuelve el valor primitivo de un objeto <code>Boolean</code> .

Algunos ejemplos de uso:

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

Para saber más

[Más información y ejemplos sobre el objeto Boolean.](#)

Autoevaluación

¿Para usar un objeto `Math` deberemos instanciarlo antes de poder usarlo?

- ☐ Sí.
- ☐ No.

No es correcto porque este objeto no dispone de constructor, con lo que para usarlo lo llamaremos directamente seguido de su propiedad o método.

Es correcto porque este objeto no dispone de constructor con lo que no podremos crear instancias del mismo.

Solución

1. Incorrecto
2. Opción correcta

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 del objeto Date

Propiedad	Descripción
constructor	Devuelve la función que creó el objeto <code>Date</code> .
prototype	Te permitirá añadir propiedades y métodos a un objeto.

Métodos del objeto Date

<code>getDate()</code>	Devuelve el día del mes (de 1-31).
<code>getDay()</code>	Devuelve el día de la semana (de 0-6).
<code>getFullYear()</code>	Devuelve el año (4 dígitos).
<code>getHours()</code>	Devuelve la hora (de 0-23).
<code>getMilliseconds()</code>	Devuelve los milisegundos (de 0-999).
<code>getMinutes()</code>	Devuelve los minutos (de 0-59).
<code>getMonth()</code>	Devuelve el mes (de 0-11).
<code>getSeconds()</code>	Devuelve los segundos (de 0-59).
<code>getTime()</code>	Devuelve los milisegundos desde media noche del 1 de Enero de 1970.
<code>getTimezoneOffset()</code>	Devuelve la diferencia de tiempo entre GMT y la hora local, en minutos.
<code>getUTCDate()</code>	Devuelve el día del mes en base a la hora UTC (de 1-31).
<code>getUTCDay()</code>	Devuelve el día de la semana en base a la hora UTC (de 0-6).
<code>getUTCFullYear()</code>	Devuelve el año en base a la hora UTC (4 dígitos).
<code>setDate()</code>	Ajusta el día del mes del objeto (de 1-31).
<code>setFullYear()</code>	Ajusta el año del objeto (4 dígitos).
<code>setHours()</code>	Ajusta la hora del objeto (de 0-23).

Algunos ejemplos de uso:

```
var d = new Date();
document.write(d.getFullYear());
document.write(d.getMonth());
document.write(d.getUTCDay());
```






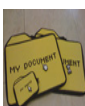

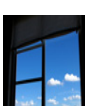
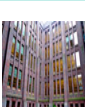
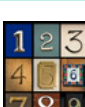
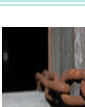
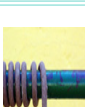




```
var d2 = new Date(5,28,2011,22,58,00);  
d2.setMonth(0);  
d.setFullYear(2020);
```

Para saber más

[Más ejemplos e información del objeto date](#)

Anexo.- Licencia recursos.

Licencias de recursos utilizados en la Unidad de Trabajo.

Recurso (1)	Datos del recurso (1)	Recurso (2)	Datos
	Autoría: Stockbyte. Licencia: Uso educativo no comercial para plataformas públicas de Formación Profesional a distancia. Procedencia: CD-DVD Num. IE008.		Autoría: Dominic's pics. Licencia: CC BY 2.0. Procedencia: http://www.flickr.com/photos/dominic
	Autoría: quinn.any. Licencia: CC BY-SA 2.0. Procedencia: http://www.flickr.com/photos/quinnanya/4399115213/sizes/z/in/photostream/		Autoría: chrisdlugosz. Licencia: CC BY 2.0. Procedencia: http://www.flickr.com/photos/chrisdlugosz
	Autoría: Caitlinator. Licencia: CC BY 2.0. Procedencia: http://www.flickr.com/photos/caitlinator/4229768048/sizes/z/in/photostream/		Autoría: infimity. Licencia: CC BY-NC-SA 2.0. Procedencia: http://www.flickr.com/photos/maxlii
	Autoría: ricardodiaz11. Licencia: CC BY 2.0. Procedencia: http://www.flickr.com/photos/ricardodiaz/3208220314/sizes/z/in/photostream/		Autoría: toxickore. Licencia: CC BY 2.0. Procedencia: http://www.flickr.com/photos/toxickore
	Autoría: zoetnet. Licencia: CC BY 2.0. Procedencia: http://www.flickr.com/photos/zoetnet/5063686922/sizes/z/in/photostream/		Autoría: pdjsphotos. Licencia: CC BY-NC-ND 2.0. Procedencia: http://www.flickr.com/photos/pdjsphotos
	Autoría: faccig. Licencia: CC BY-SA 2.0. Procedencia: http://www.flickr.com/photos/faccig/3579566706/sizes/z/in/photostream/		Autoría: Josep Ma. Rosell. Licencia: CC BY 2.0. Procedencia: http://www.flickr.com/photos/bateg
	Autoría: conskeptical. Licencia: CC BY-SA 2.0. Procedencia: http://www.flickr.com/photos/conskeptical/361555297/sizes/z/in/photostream/		Autoría: Darwin Bell. Licencia: CC BY 2.0. Procedencia: http://www.flickr.com/photos/darwinbell
	Autoría: Cuito Cuanavale. Licencia: CC BY 2.0. Procedencia: http://www.flickr.com/photos/hexadecimal_time/2254800793/sizes/z/in/photostream/		Autoría: Robbert van der Steeg. Licencia: CC BY-SA 2.0. Procedencia: http://www.flickr.com/photos/robbert