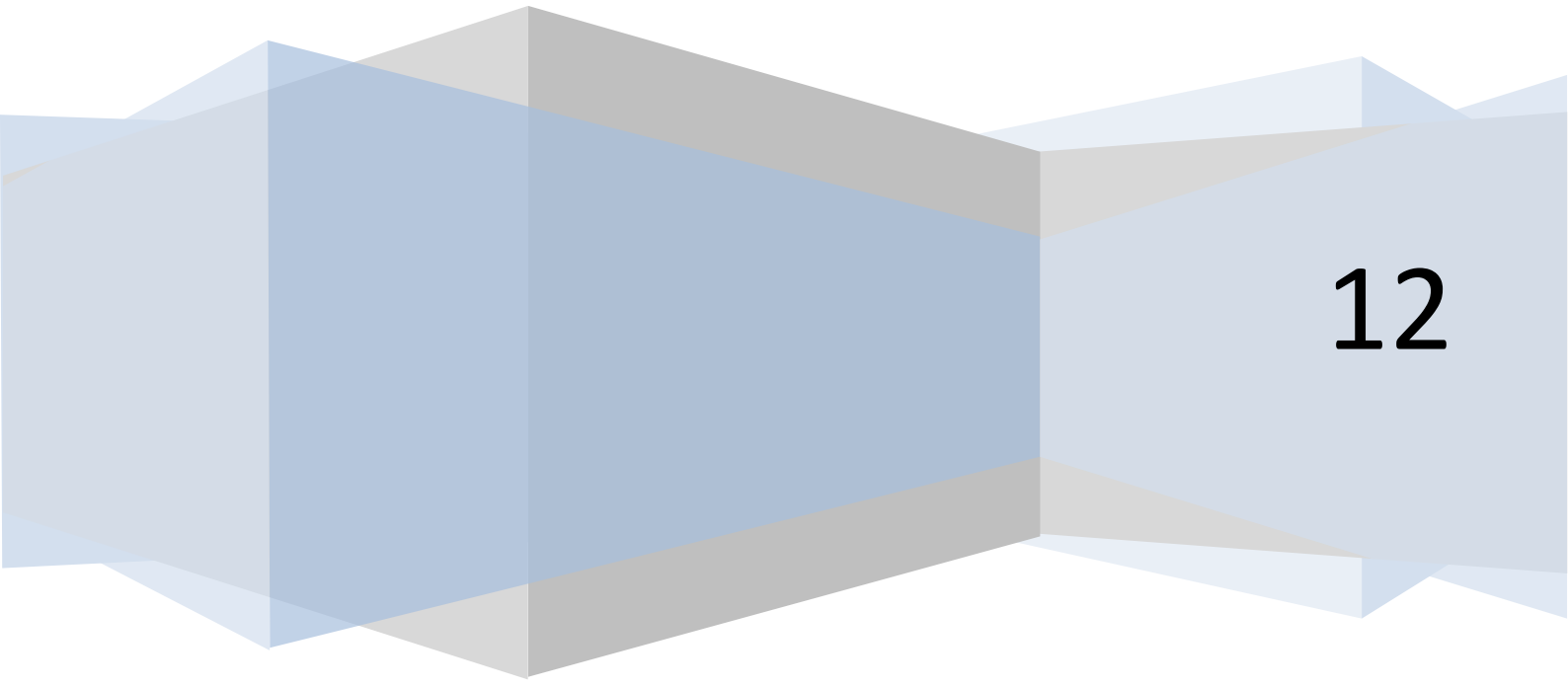
DESARROLLO WEB ENTORNO CLIENTE

Desarrollo de Aplicaciones Web

José Luis Comesaña



12

ÍNDICE

Arquitecturas y lenguajes de programación en clientes web.

*La empresa "**BK Programación**", ha sido informada a través de uno de sus asesores, de que se ha abierto el plazo, para concursar a la adjudicación de un proyecto de modernización de la web de una importante empresa dedicada a la moda.*
***Ada**, la directora de "BK Programación", decide que es una buena oportunidad para conseguir trabajo para su empresa, por lo que consultan el pliego de requisitos exigidos y ve que su empresa tiene personal suficiente para dar solución a dicho proyecto. Se envía la solicitud y dos meses más tarde, la empresa de Ada sale adjudicataria del contrato de modernización.*
El objetivo principal del proyecto es el de dar un mayor dinamismo a las páginas y actualizar la web a lo que se conoce como Web 2.0 (mayor interacción, interoperabilidad, aplicaciones más ricas y no intrusivas, etc.)
Ada considera que Antonio (estudiante de ciclo y trabajador de su empresa), con conocimientos de lenguaje HTML, podría formarse en las técnicas necesarias para realizar dicho trabajo de actualización y como además el proyecto tendrá un plazo máximo de entrega de 1 año, dispondrá de tiempo más que suficiente para ello.
Ada, informa a sus trabajadores de la adjudicación de dicho contrato, y Antonio (bajo la tutoría de Juan), decide comenzar a investigar en las diferentes opciones disponibles para llevar a cabo su trabajo.

1.- Desarrollo web.

Caso Práctico
***Juan** (técnico en Desarrollo de Aplicaciones Informáticas), será el tutor de Antonio en la empresa BK Programación y le irá guiando durante todo el proceso de formación y en la realización del proyecto de modernización de la web de la compañía de moda.*
Para comenzar, Juan le recomienda a Antonio una pequeña introducción sobre lo que es el Desarrollo Web y las diferentes áreas que abarca el Diseño Web.

La web fue inicialmente concebida y creada por **Tim Berners-Lee**, un especialista del laboratorio europeo de partículas (CERN) en 1989. En sus mismas palabras, había una "necesidad de una herramienta colaborativa que soportara el conocimiento científico" en un contexto internacional. Él y su compañero Robert Cailliau crearon un prototipo web para el CERN y lo mostraron a la comunidad para sus pruebas y comentarios.

Dicho prototipo estaba basado en el concepto de hipertexto (*Texto que cuando pulsamos en él nos conduce a otro texto, objeto, sonido, video, sección o documento relacionado*). Como resultado se crearon unos protocolos (*cuando pulsamos en él nos conduce a otro texto, objeto, sonido, video, sección o documento relacionado*) y especificaciones que han sido adoptados universalmente e incorporados a Internet, gracias a aportaciones posteriores como el desarrollo por la NCSA de la popular interfaz MOSAIC.

Todos los prototipos y desarrollos posteriores crecieron bajo la guía del **consorcio W3C**, que es una organización con base en el MIT de Massachusetts y que se responsabiliza de desarrollar y mantener los estándares web.

Por **Web se pueden entender tres cosas distintas:** el proyecto inicial del CERN, el conjunto de protocolos desarrollados en dicho proyecto o bien el espacio de información formado por todos los servidores interconectados. Cuando se habla de la Web generalmente se hace referencia a esto último.

Muchas de las discusiones sobre Diseño Web o Desarrollo Web son confusas, ya que la expresión varía considerablemente. Mientras que la mayoría de la gente tiene algún tipo de noción sobre **lo**

que significa Diseño Web, solamente unos pocos son capaces de definirlo con exactitud, y tú vas a estar dentro de ese grupo.

Algunos componentes como diseño gráfico o programación, forman parte de esa discusión, pero su importancia en la construcción de webs varía de persona a persona y de web a web. Algunos consideran la creación y organización de contenido - o más formalmente, la arquitectura de la información - como el aspecto más importante del Diseño Web. Otros factores como - la facilidad de uso, el valor y funcionalidad del sitio web en la organización, su funcionalidad, accesibilidad, publicidad, etc. también forman una parte muy activa hoy en día sobre lo que se considera Diseño Web.

El Desarrollo Web ha sido y sigue estando muy influenciado por múltiples campos como el de las nuevas tecnologías, los avances científicos, el diseño gráfico, la programación, las redes, el diseño de interfaces (*Medio o forma a través de la cuál un usuario se comunica con el ordenador*) de usuario, la usabilidad y una variedad de múltiples recursos. Por lo tanto el Desarrollo Web es realmente un campo multidisciplinar.

¿Cuando nos referimos a la Web estamos hablando de?

- ○ Proyecto del CERN.
- ○ Protocolos utilizados en el proyecto del CERN.
- ◉ **Espacio de información que nos proporcionan los servidores a través de Internet.**

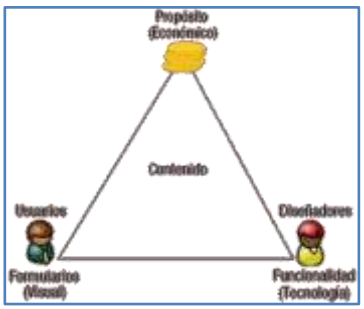
Efectivamente es correcta, ya que hoy en día cuando hablamos de Web estamos hablando de todo ese entramado de información proporcionado por los servidores de información interconectados.

1.1.- Áreas

Hay cinco áreas que cubren la mayor parte de las facetas del Diseño Web:

- ✓ **Contenido**: incluye la forma y organización del contenido del sitio. Esto puede abarcar desde cómo se escribe el texto hasta cómo está organizado, presentado y estructurado usando tecnologías de marcas como HTML.
- ✓ **Visual**: hace referencia a la plantilla empleada en un sitio web. Esta plantilla generalmente se genera usando HTML, CSS o incluso Flash y puede incluir elementos gráficos para decoración o para navegación. El aspecto visual es el aspecto más obvio del Diseño Web, pero no es la única disciplina o la más importante.
- ✓ **Tecnología**: aunque muchas de las tecnologías web como HTML o CSS entran dentro de esta categoría, la tecnología en este contexto generalmente hace referencia a los diferentes tipos de elementos interactivos de un sitio web, generalmente aquellos construidos empleando técnicas de programación.
- ✓ **Distribución**: la velocidad y fiabilidad con la que un sitio web se distribuye en Internet o en una red interna corporativa está relacionado con el hardware/software utilizado y el tipo de arquitectura de red utilizada en la conexión.
- ✓ **Propósito**: la razón por la que un sitio web existe, generalmente está relacionada con algún aspecto de tipo económico. Por lo tanto este elemento debería considerarse en todas las decisiones que tomemos en las diferentes áreas.

El porcentaje de influencia de cada una de estas áreas en un sitio web, puede variar dependiendo del tipo de sitio que se está construyendo. Una página personal generalmente no tiene las consideraciones económicas que tendría una-web que va a vender productos en Internet.



Una forma de pensar en los componentes del Diseño Web es a través de la metáfora de la pirámide mostrada en la figura de la derecha. El contenido proporciona los ladrillos que formarán la pirámide, pero la base de la pirámide se fundamenta tanto en la parte visual como en la parte tecnológica y con el punto de vista económico puesto como

objetivo o propósito final en la mayoría de los casos.
Aunque la analogía de la pirámide es una forma un poco abstracta de describir el Diseño Web, es una herramienta que nos permite visualizar la interrelación de los diferentes componentes de la construcción Web.

Hoy en día los sitios web siguen un modelo basado en la **programación cliente-servidor** con tres **elementos** comunes:

- ✓ El lado del **servidor(server-side)**: incluye el hardware y software del servidor Web así como diferentes elementos de programación y tecnologías incrustadas. Las tecnologías pueden abarcar un rango amplio desde programasCGI escritos en PERL hasta aplicaciones multihilo (*También denominado multiproceso hace referencia a la posibilidad de ejecutar diferentes trozos de código de una misma aplicación de forma simultánea*) basadas en Java, incluyendo tecnologías de servidor de bases de datos que soporten múltiples sitios web.
- ✓ El lado del **cliente(client-side)**: este elemento hace referencia a los navegadores web y está soportado por tecnologías como HTML, CSS y lenguajes como JavaScript y controles ActiveX, los cuales se utilizan para crear la presentación de la página o proporcionar características interactivas. Es justamente aquí dónde nos vamos a centrar a lo largo de todo el módulo.
- ✓ La **red**: describe los diferentes elementos de conectividad (*Capacidad que tiene un dispositivo para poder conectarse a otros. Aquí se detallan los diferentes protocolos y material utilizado para poder realizar dicha conexión*) utilizados para mostrar el sitio web al usuario.

El entendimiento completo de todos los aspectos técnicos del medio Web, incluyendo la componente de red, es de vital importancia para llegar a ser un buen Diseñador Web.

2.- Lenguajes de programación en clientes web.

Caso práctico
El proyecto de modernización de la empresa de moda va a consistir, principalmente, en añadir mayor funcionalidad a las páginas ya existentes, más control en los formularios, dinamismo, modernización de la mecánica de varios procesos, etc.
***Antonio** tendrá que centrarse en los lenguajes de programación en el lado del cliente, ya que éstos son los que le ayudarán a aportar las capacidades solicitadas en el proyecto.*
Antonio se pone manos a la obra y consulta bajo la tutoría de Juan, qué opciones tienen para llevar a cabo esa programación y analizan las características, compatibilidades y seguridad aportada por los lenguajes de programación en el lado del cliente.

Cuando hablamos de tecnologías empleadas en lenguajes de programación web podemos citar dos grupos básicos: **client-side** y **server-side**. Las tecnologías client-side son aquellas que son ejecutadas en el cliente, generalmente en el contexto del navegador web. Cuando los programas o tecnologías son ejecutadas o interpretadas por el servidor estamos hablando de programación server-side.

Uno de los objetivos en la programación web es saber **escoger la tecnología correcta** para tu trabajo. Muchas veces los desarrolladores escogen rápidamente una tecnología favorita, que puede ser JavaScript, ColdFusion o PHP y la usan en todas las situaciones. La realidad es que cada tecnología tiene sus pros y sus contras. En general las tecnologías client-side y server-side poseen características que las hacen complementarias más que adversarias. Por ejemplo, cuando añadimos un formulario para recoger información y grabarla en una base de datos, es obvio que tendría más sentido chequear el formulario en el lado del cliente para asegurarnos que la información introducida es correcta, justo antes de enviar la información a la base de datos del servidor. La programación en el lado del cliente consigue que la validación del formulario sea mucho más efectiva y que el usuario se sienta menos frustrado al cubrir los datos en el formulario. Por otro lado el almacenar los datos en el servidor estaría mucho mejor gestionado por una tecnología del lado del servidor (server-side), dando por supuesto que la base de datos estará en el lado del servidor.

Cada tipo general de programación tiene su propio lugar y la mezcla es generalmente la mejor solución. Cuando hablamos de lenguajes de programación en clientes web, podemos distinguir dos variantes:

- ✓ Lenguajes que nos permiten dar formato y estilo a una página web (HTML, CSS, etc.).
- ✓ Lenguajes que nos permite aportar dinamismo a páginas web (lenguajes de scripting).

En este módulo nos vamos a centrar principalmente en estos últimos, los lenguajes de scripting, y en particular en el lenguaje **JavaScript** que será el lenguaje que utilizaremos a lo largo de todo este módulo formativo.

Tabla comparativa de lenguajes de programación web cliente – servidor.

Lado del Cliente (client-side)	Lado del servidor (server-side)
✓ Aplicaciones de Ayuda. ✓ Programas del API del navegador. ➔ Plug-ins de Netscape. ➔ Controles ActiveX. ➔ Applets de Java. ✓ Lenguajes de scripting. ➔ JavaScript. ➔ VBScript.	✓ Scripts y programas CGI. ✓ Programas API del servidor. ➔ Módulos de Apache. ➔ Extensiones ISAPI y filtros. ➔ Servlets de Java. ✓ Lenguajes de scripting. ➔ PHP. ➔ Active Server Pages (ASP/ASP.NET). ➔ ColdFusion.
...	

Hemos escogido JavaScript porque es el lenguaje de script (*Lenguaje de guiones o lenguaje de órdenes que se almacena por lo general en archivos de texto plano y que será ejecutado por un programa intérprete*) más utilizado en la programación en el lado del cliente, y está soportado mayoritariamente por todas las plataformas (*Sistema operativo utilizado por un determinado dispositivo*). Por lo tanto a partir de ahora todas las referencias que hagamos estarán enfocadas hacia JavaScript.

A continuación te mostramos un esquema de las **4 capas del desarrollo web** en el lado del cliente, en la que se puede ver que JavaScript se sitúa en la capa superior gestionando el comportamiento de la página web.

Tabla de las 4 capas del desarrollo web en el lado del cliente.	
Comportamiento (JavaScript)	
Presentación (CSS)	
Estructura (DOM / estructura HTML)	Contenido Estructurado (documento HTML)
Contenido (texto, imágenes, vídeos, etc.)	

2.1.- Características.

Icono indicador de actividad AJAX.

Como vimos anteriormente, los lenguajes de programación para clientes web no son un reemplazo de la programación en el lado del servidor. Cualquier web que reaccione dinámicamente a interacciones del usuario o que almacene datos, estará gestionada por lenguajes de script en el lado del servidor, incluso aunque usemos JavaScript en el cliente para mejorar la experiencia de usuario. Las razones son simples:

- ✓ **Primero**: JavaScript por sí mismo no puede escribir ficheros en el servidor. Puede ayudar al usuario a elegir opciones o preparar datos para su envío, pero después de eso solamente podrá ceder los datos al lenguaje de servidor encargado de la actualización de datos.
- ✓ **Segundo**: no todos los clientes web ejecutan JavaScript. Algunos lectores, dispositivos móviles, buscadores, o navegadores instalados en ciertos contextos están entre aquellos que no pueden realizar llamadas a JavaScript, o que simplemente son incompatibles con el código de JavaScript que reciben. Aunque ésto ocurra, nuestra página web debería ser completamente funcional con JavaScript desactivado. Utilizaremos JavaScript para conseguir que la experiencia de navegación web sea lo más rápida, moderna o divertida posible, pero no dejaremos que nuestra web deje de funcionar si JavaScript no está funcionando.
- ✓ **Tercero**: uno de los caminos que más ha integrado la programación cliente con la programación servidor ha surgido gracias a **AJAX**. El proceso "asíncrono" de AJAX se ejecuta en el navegador del cliente y emplea JavaScript. Este proceso se encarga de solicitar datos XML, o enviar datos al lenguaje de servidor y todo ello de forma transparente en background. Los datos devueltos por el servidor pueden ser examinados por JavaScript en el lado del cliente, para actualizar secciones o partes de la página web. Es así como funciona una de las webs más populares en Internet, el servicio de Google Maps (http://maps.gogle.com).

JavaScript está orientado a dar soluciones a:

- ✓ Conseguir que nuestra página web responda o reaccione directamente a la interacción del usuario con elementos de formulario y enlaces hipertexto.
- ✓ La distribución de pequeños grupos de datos y proporcionar una interfaz amigable para esos datos.
- ✓ Controlar múltiples ventanas o marcos de navegación, plug-ins, o applets Java basados en las elecciones que ha hecho el usuario en el documento HTML.
- ✓ Pre-procesar datos en el cliente antes de enviarlos al servidor.
- ✓ Modificar estilos y contenido en los navegadores de forma dinámica e instantáneamente, en respuesta a interacciones del usuario.
- ✓ Solicitar ficheros del servidor, y enviar solicitudes de lectura y escritura a los lenguajes de servidor.

Los lenguajes de script como JavaScript no se usan solamente en las páginas web. Los intérpretes (*Programa informático capaz de analizar y ejecutar código escrito en un lenguaje de programación de alto nivel (aquel que se acerca más a la capacidad cognitiva de las personas en lugar de a la capacidad ejecutora de la máquina)*) de JavaScript están integrados en múltiples aplicaciones de uso cotidiano. Estas aplicaciones proporcionan su propio modelo de acceso y gestión de los módulos que componen la aplicación y para ello comparten el lenguaje JavaScript en cada aplicación. Podríamos citar varios ejemplos como: Google Desktop Gadgets, Adobe Acrobat, Dreamweaver, OpenOffice.org, Google Docs, etc.

¿Podríamos escribir con JavaScript un fichero de texto directamente en el servidor?

- ○ Sí, pero sólo si se trata de una cookie.
- ◉ **De ninguna manera.**
- ○ Sí, empleando AJAX.

JavaScript es un lenguaje que interpreta el navegador web y no es un lenguaje de servidor, por lo que no tendrá acceso a los recursos para poder escribir en él.

2.2.- Compatibilidades.

A diferencia de otros tipos de scripts como los CGI, JavaScript es interpretado por el cliente. Actualmente existen múltiples clientes o navegadores que soportan JavaScript, incluyendo Firefox, Google Chrome, Safari, Opera, Internet Explorer, etc. Por lo tanto, cuando escribimos un script en nuestra página web, tenemos que estar seguros de que será interpretado por diferentes navegadores y que aporte la misma funcionalidad y características en cada uno de ellos. Ésta es otra de las diferencias con los scripts de servidor en los que nosotros dispondremos del control total sobre su interpretación.

Cada tipo de navegador da soporte a diferentes características del JavaScript y además también añaden sus propios **bugs o fallos**. Algunos de estos fallos son específicos de la plataforma sobre la que se ejecuta ese navegador, mientras que otros son específicos del propio navegador en sí.

Navegadores actuales y su soporte para distintas tecnologías									
Browser	JavaScript	ECMAScript 3	DOM 1	DOM 2	DOM 3	XPath	DHTML	XMLHttpRequest	Rich editing
Amaya	No [note 1]	No [note 1]	No [note 1]	No	No	No	No	No	No
AOL Explorer	Yes	Yes	Partial	Yes	No	No	Yes	Yes	Yes
Avant	Yes	Yes	Partial	No	No	No	Yes	Yes	Yes
Camino	Yes	Yes	Yes	Yes	No [note 2]	Yes	Yes	Yes	Yes
Dillo	No	No	No	No	No	No	No	No	No
DocZilla	Yes	No	Yes	Yes	Partial	Yes	Yes	Yes	No
ELinks	Partial	Partial	No	No	No	No	No	No	No
Web	Yes	Yes	Yes	Yes	No [note 2]	Yes	Yes	Yes	Yes
Flock	Yes	Yes	Yes	Yes	No [note 2]	Yes	Yes	Yes	Yes
Galeon	Yes	Yes	Yes	Yes	No [note 2]	Yes	Yes	Yes	Yes
Google Chrome	Yes	Yes	Yes	Yes	Partial	Yes	Yes	Yes	Yes
iCab	Yes	Yes	Partial	Partial	No	No	Yes	Yes	No
Internet Explorer	Yes	Yes	Yes	Yes [note 3]	Partial	Yes	Yes	Yes	Yes
Internet Explorer for Mac	Yes	Yes	Partial	No	No	No	Yes	No	No
K-Meleon	Yes	Yes	Yes	Yes	No [note 2]	Yes	Yes	Yes	Yes
Konqueror	Yes	Yes	Yes	Yes	Partial	No	Yes	Yes	No
Links	No [note 4]	No	No	No	No	No	No	No	No
Lynx	No	No	No	No	No	No	No	No	No
Maxthon	Yes	Yes	Partial	No	No	Yes	Yes	Yes	Yes
Midori	Yes	Yes	Yes	Yes	Partial [note 5]	Yes [note 5]	Yes	Yes	Yes
Mosaic	No	No	No	No	No	No	No	No	No
Mozilla	Yes	Yes	Yes	Yes	No [note 2]	Yes	Yes	Yes	Yes
Mozilla Firefox	Yes	Yes	Yes	Yes	Partial	Yes	Yes	Yes	Yes

Navegadores actuales y su soporte para distintas tecnologías									
Browser	JavaScript	ECMAScript 3	DOM 1	DOM 2	DOM 3	XPath	DHTML	XMLHttpRequest	Rich editing
					[note 2]				
Netscape	Yes	Yes	Yes	Yes	No [note 2]	Yes	Yes	Yes	Yes
Netscape Browser	Yes	Yes	Depends [18]	Depends [18]	No [note 2]	Depends [18]	Yes	Yes	Yes
Netscape Navigator	Yes	Partial	No	No	No	No	Yes	No	No
Netscape Navigator 9	Yes	Yes	Yes	Yes	No [note 2]	Yes	Yes	Yes	Yes
NetSurf	No	No	No	No	No	No	No	No	No
OmniWeb	Yes	Yes	Yes	Yes	No	No	Yes	Yes	No
Opera	Yes	Yes	Yes	Yes	Partial	Yes	Yes	Yes	Yes
Safari	Yes	Yes	Yes	Yes	Partial [note 5]	Yes	Yes	Yes	Yes
SeaMonkey	Yes	Yes	Yes	Yes	No [note 2]	Yes	Yes	Yes	Yes
Shiira	Yes	Yes	Yes	Yes	No	No	Yes	Yes	Yes
Sleipnir	Yes	Yes	Partial	No [note 3]	No	Yes	Yes	Yes	Yes
WorldWideWeb	No	No	No	No	No	No	No	No	No
w3m	No	No	No	No	No	No	No	No	No
Browser	JavaScript	ECMAScript 3	DOM 1	DOM 2	DOM 3	XPath	DHTML	XMLHttpRequest	Rich editing

A veces las incompatibilidades entre navegadores al interpretar el código de JavaScript no vienen dadas por el propio código en si, sino que su origen proviene del código fuente HTML. Por lo tanto es muy importante que tu código HTML siga las **especificaciones del estándar W3C** y para ello dispones de herramientas como el validador HTML W3C:

http://validator.w3.org/

También tienes que tener precaución con las **limitaciones en el uso de JavaScript**:

- ✓ No todos los navegadores soportan lenguajes de script (en especial JavaScript) en el lado del cliente.
- ✓ Algunos dispositivos móviles tampoco podrán ejecutar JavaScript.
- ✓ Incluso las implementaciones más importantes de JavaScript en los diferentes navegadores no son totalmente compatibles entre ellas: por ejemplo diferentes incompatibilidades entre Firefox e Internet Explorer.
- ✓ La ejecución de código JavaScript en el cliente podría ser desactivada por el usuario de forma manual, con lo que no podremos tener una confianza ciega en que se vaya a ejecutar siempre tu código de JavaScript.
- ✓ Algunos navegadores de voz, no interpretan el código de JavaScript.

Si te apetece puedes realizar el siguiente TEST INTERACTIVO en el que podrás comprobar tus conocimientos sobre las áreas de desarrollo del Diseño Web y los tipos de lenguajes client-side y server-side:

Escribe el nombre de cada Área del Diseño Web identificada en los enunciados.

Incluye la forma y organización del contenido del sitio.	contenido
Referencia a los diferentes tipos de elementos interactivos de un sitio web.	tecnología
Hace referencia a las plantillas usadas en el sitio web.	visual
La principal razón por la que un sitio web existe.	propósito
Hace referencia a la velocidad y fiabilidad de una página web.	distribución

Agrupa los siguientes lenguajes en el grupo que corresponda

client-side	server-side
HTML, JavaScript, CSS	PHP, ASP, Perl CGI

2.3.- Seguridad.

JavaScript proporciona un gran potencial para diseñadores maliciosos que quieran distribuir sus scripts a través de la web. Para evitar esto, los navegadores web en el cliente aplican dos tipos de restricciones:

- ✓ Por razones de seguridad cuando se ejecuta código de JavaScript éste lo hace en un "espacio seguro de ejecución" en el cuál solamente podrá realizar tareas relacionadas con la web, nada de tareas genéricas de programación como creación de ficheros, etc.
- ✓ Además los scripts están restringidos por la política de "mismo origen": la cual quiere decir que los scripts de una web no tendrán acceso a información tal como usuarios, contraseñas, o cookies enviadas desde otra web. La mayor parte de los agujeros de seguridad son infracciones tanto de la **política** de "**mismo origen**" como de la política de "espacio seguro de ejecución".

Al mismo tiempo es importante entender las **limitaciones que tiene JavaScript** y que, en parte, refuerzan sus capacidades de seguridad. JavaScript no podrá realizar ninguna de las siguientes tareas:

- ✓ Modificar o acceder a las preferencias del navegador del cliente, las características de apariencia de la ventana principal de navegación, las capacidades de impresión, o a los botones de acciones del navegador.
- ✓ Lanzar la ejecución de una aplicación en el ordenador del cliente.
- ✓ Leer o escribir ficheros o directorios en el ordenador del cliente (con la excepción de las cookies).
- ✓ Escribir directamente ficheros en el servidor.
- ✓ Capturar los datos procedentes de una transmisión en streaming de un servidor, para su retransmisión.
- ✓ Enviar e-mails a nosotros mismos de forma invisible sobre los visitantes a nuestra página web (aunque sí que podría enviar datos a una aplicación en el lado del servidor capaz de enviar correos).
- ✓ Interactuar directamente con los lenguajes de servidor.
- ✓ Las páginas web almacenadas en diferentes dominios no pueden ser accesibles por JavaScript.
- ✓ JavaScript es incapaz de proteger el origen de las imágenes de nuestra página.
- ✓ Implementar multiprocesamiento o multitarea.
- ✓ Otro tipo de **vulnerabilidades** que podemos encontrar están relacionadas con el XSS. Este tipo de vulnerabilidad viola la política de "mismo origen" y ocurre cuando un atacante es capaz de inyectar código malicioso en la página web presentada a su víctima. Este código malicioso puede provenir de la base de datos a la cual está accediendo esa víctima. Generalmente este tipo de errores se deben a fallos de implementación de los programadores de navegadores web.

Otro aspecto muy relacionado con la seguridad son los defectos o imperfecciones de los navegadores web o plugins utilizados. Estas imperfecciones pueden ser empleadas por los atacantes para escribir scripts maliciosos que se puedan ejecutar en el sistema operativo del usuario.

El **motor de ejecución de JavaScript** es el encargado de ejecutar el código de JavaScript en el navegador y por lo tanto es en él dónde recaerá el peso fuerte de la implementación de la seguridad. Podríamos citar varios ejemplos de motores de JavaScript:

- ✓ Active Script de Microsoft: tecnología que soporta JScript como lenguaje de scripting. A menudo se considera compatible con JavaScript, pero Microsoft emplea múltiples características que no siguen los estándares ECMA.
- ✓ El kit de herramientas Qt C++ también incluye un módulo intérprete de JavaScript.
- ✓ El lenguaje de programación Java en su versión JDK 1.6 introduce un paquete denominada javax.script que permite la ejecución de JavaScript.
- ✓ Y por supuesto todos los motores implementados por los navegadores web como Mozilla, Google, Opera, Safari, etc. Cada uno de ellos da soporte a alguna de las diferentes versiones de JavaScript.

Hoy en día una de las características que más se resalta y que permite diferenciar a unos navegadores de otros, es la rapidez con la que sus motores de JavaScript pueden ejecutar las aplicaciones, y la seguridad y aislamiento que ofrecen en la ejecución de las aplicaciones en diferentes ventanas o pestañas de navegación.

3.- Herramientas y utilidades de programación (I).

Caso práctico
En BK Programación Juan y Antonio han decidido, después de analizar los requisitos del proyecto y de consultarlo con su directora Ada, que utilizarán el lenguaje JavaScript para la realización del proyecto de modernización de la empresa de moda.
Lo primero que tienen que hacer es decidir qué tipo de herramientas y utilidades adicionales van a usar para realizar la programación con el lenguaje JavaScript. Buscan alguna herramienta que les permita introducir el código de JavaScript fácilmente y les aporte ayudas adicionales detectando errores sintácticos, partes incompletas, etc.
También tienen qué decidir qué navegador o navegadores van a usar para comprobar la ejecución y compatibilidad de su código de JavaScript.

La mejor forma de aprender JavaScript es tecleando el código HTML y JavaScript en un simple documento de texto. La elección del editor depende de ti, pero aquí te voy a dar algunas pistas para realizar una buena elección.

Para aprender JavaScript no se recomiendan editores del estilo **WYSIWYG** (What You See is What You Get) como Dreamweaver o FrontPage, ya que estas herramientas están más orientadas a la modificación de contenido y presentación, y nosotros nos vamos a centrar más en el código fuente de la página.

Uno de los **factores** importantes que tienes que tener en cuenta a la hora de elegir un editor, es ver la facilidad con la que se pueden grabar los ficheros con extensión .html. Independientemente del sistema operativo que estés utilizando cualquier programa que te permita grabar ficheros directamente con la extensión .htm o .html te evitaría un gran número de problemas. También hay que tener en cuenta la codificación que emplea ese programa para grabar los ficheros. Por ejemplo en Microsoft Word cuando intentamos guardar archivos éste intenta almacenarlos en un formato binario de Word - algo que los navegadores web no pueden cargar. Para grabar ese archivo con extensión .txt o .html requiere moverse un poco más por el menú de diálogo "Guardar como", lo cuál es realmente una clara desventaja.

La idea es decantarse por editores que posean **características** que te faciliten la programación web, como por ejemplo:
- ✓ Sintaxis con codificación de colores. Que resalte automáticamente en diferente color o tipos de letra los elementos del lenguaje tales como objetos, comentarios, funciones, variables, etc.
- ✓ Verificación de sintaxis. Que te marque los errores en la sintaxis del código que estás escribiendo.
- ✓ Que te permita diferenciar los comentarios del resto del código.
- ✓ Que genere automáticamente partes del código tales como bloques, estructuras, etc.
- ✓ Que disponga de utilidades adicionales, tales como cliente FTP para enviar tus ficheros automáticamente al servidor, etc.

Dependiendo del sistema operativo que utilices en tu ordenador dispones de múltiples **opciones de editores**. Cada una es perfectamente válida para poder programar en JavaScript. Algunos ejemplos de editores web gratuitos son:
- ✓ Para Windows tienes: Notepad++, Aptana Studio, Bluefish, Eclipse, NetBeans, etc.
- ✓ Para Macintosh tienes: Aptana Studio, Bluefish, Eclipse, KompoZer, Nvu, etc.
- ✓ Para Linux tienes: KompoZer, Amaya, Quanta Plus, Bluefish, codetch, etc.

Otro de los componentes obligatorio para aprender JavaScript es el navegador web. No es necesario que te conectes a Internet para comprobar tus scripts realizados con JavaScript. Puedes realizar dicha tarea sin conexión. Ésto quiere decir que puedes aprender JavaScript y crear aplicaciones con un ordenador portátil y desde cualquier lugar sin necesitar Internet para ello.

Si te apetece puedes realizar el siguiente **TEST** en el que podrás comprobar tus conocimientos sobre las áreas de desarrollo del Diseño Web y los tipos de lenguajes client-side y server-side:

Busca 6 Editores que te permitan programar en JavaScript

Z	G	J	M	Q	T	B	Z	A	M	I	V
F	H	K	B	S	U	Y	Q	C	L	N	H
V	B	K	M	P	E	C	L	I	P	S	E
A	P	T	U	E	Y	I	Q	D	I	S	B
P	X	H	U	P	W	M	X	F	B	C	A
T	J	T	I	N	O	T	E	P	A	D	E
A	E	R	T	Y	E	U	C	S	Z	X	G
N	C	V	B	N	L	T	I	A	R	K	J
A	Z	F	G	B	K	H	B	N	T	T	O
F	T	Y	U	I	J	K	P	E	S	R	W
K	O	M	P	O	Z	E	R	J	A	H	B
W	S	X	C	V	G	H	R	U	B	N	Y
F	G	H	J	K	L	R	T	F	I	V	S

Busca el nombre de 5 navegadores web

Z	G	J	M	Q	T	B	Z	A	M	I	V
F	S	A	F	A	R	I	Q	C	L	N	H
V	B	K	M	P	E	C	L	I	P	X	E
A	P	T	E	E	Y	I	Q	D	O	S	B
P	X	H	U	M	W	M	X	F	B	C	A
T	J	T	I	N	O	P	E	R	A	D	E
A	E	R	T	Y	E	R	C	S	Z	X	G
N	C	V	B	N	I	T	H	A	R	K	J
A	Z	F	G	F	K	H	B	C	T	T	O
F	T	Y	U	I	J	K	P	E	S	R	W
R	E	X	P	L	O	R	E	R	A	H	B
W	S	X	C	V	G	H	R	U	B	N	Y
F	G	H	J	K	L	R	T	F	I	V	S

3.1.- Herramientas y utilidades de programación (II).

El tipo de **navegador web** que utilices es elección tuya. Eso sí, te recomiendo que uses las últimas versiones disponibles para evitar problemas de seguridad e incompatibilidades. Algunos ejemplos de navegadores gratuitos son:

- ✓ Para Windows tienes: Mozilla Firefox, Google Chrome, Safari, Opera, Internet Explorer, etc.
- ✓ Para Macintosh tienes: Mozilla Firefox, Safari, Google Chrome, Internet Explorer, etc.
- ✓ Para Linux tienes: Mozilla Firefox, Konqueror, Opera, etc.

Una recomendación muy interesante es el disponer de 2 o 3 tipos de navegadores diferentes, ya que así podrás comprobar la compatibilidad de tu página web y ver si tu código fuente deJavaScript se ejecuta correctamente en todos ellos.

Para ajustar un poco más tu entorno de trabajo, lo último que necesitas es el poder ejecutar tu editor web y tu navegador de forma simultánea, ya que el **flujo típico de trabajo en JavaScript** va a ser:

1. Introducir HTML, JavaScript y CSS en el documento original en el editor web.

2. Guardarlo en disco.
3. Cambiarte al navegador web.
4. Realizar una de las siguiente tareas:
   - ✓ Si es un nuevo documento, abrirlo a través de la opción Abrir del menú Archivo > Abrir Archivo.
   - ✓ Si el documento ya está cargado en el navegador pues simplemente recargar la página.

Los pasos 2 al 4 son acciones que se van a ejecutar muy frecuentemente. Esa secuencia grabar-cambiar-recargar la realizarás tantas veces cuando estés escribiendo y depurando tu script, que llegará a ser prácticamente un acto reflejo. Algunos editores ya disponen de teclas rápidas para realizar esta tarea. Todo ello dependerá del tipo de editor, navegador y sistema operativo que utilices.

Otro aspecto muy importante es la **validación**. Puedes ahorrarte muchas horas de comprobaciones simplemente asegurándote de que tu código HTML es válido. Si tu código HTML contiene imperfecciones, tienes muchas posibilidades de que tu JavaScript o CSS no funcionen de la manera esperada, ya que ambos dependen de los elementos HTML y sus atributos. Cuanto más te ajustes a las especificaciones del estándar, mejor resultado obtendrás entre los diferentes tipos de navegadores.

El consorcio W3C, el cuál diseño el lenguaje HTML, desarrolló un validador que te permitirá chequear si tu página web cumple las especificaciones indicadas por el elemento DOCTYPE que se incluye al principio de cada páginaweb que realices. El Validador será tu amigo y te permitirá realizar unas páginas más consistentes.

La dirección del **Validador W3C** es:
http://validator.w3.org/

Ofrece tres formas de introducción de tu código para la validación - copiando y pegando tu código en un formulario, enviando el fichero .html o bien indicando la dirección URL dónde se encuentra nuestra página web. A continuación se pulsa el botón de chequear y el validador nos devolverá los errores encontrados. Realizaremos los cambios necesarios y procederemos a validar de nuevo hasta que no tengamos más errores.

Seguramente te interesará tener mayor información sobre las diferentes especificaciones HTML y CSS.

http://www.w3.org/TR/html4/
http://www.w3.org/TR/html5/
http://www.w3.org/TR/xhtml1/

Especificación sobre:

http://www.w3.org/TR/CSS21/
http://jigsaw.w3.org/css-validator/

4.- Integración de código Javascript con HTML (I).

Caso práctico
Antonio ha decidido ya el editor y los navegadores web que va a utilizar para programar con JavaScript, así que se pone manos a la obra y mira cuales son los primeros pasos para integrar el nuevo código de JavaScript en el HTML.
Como Antonio ya conoce el lenguaje HTML, puesto que lo estudió en uno de los módulos que está cursando en su ciclo formativo, se centra en la parte específica de HTML que le permitirá integrar el nuevo lenguaje de programación JavaScript con el lenguaje HTML que ya conoce.

Ahora que ya conoces las herramientas que puedes utilizar para comenzar a programar en JavaScript, vamos a ver la forma de **integrar el código de JavaScript** en tu código HTML.

Los navegadores web te permiten varias opciones de inserción de código de JavaScript. Podremos insertar código usando las etiquetas `<script> </script>` y empleando un atributo `type` indicaremos qué tipo de lenguaje de script estamos utilizando:

Por ejemplo:

```
<script  type="text/javascript">
   //  El código de JavaScript vendrá aquí.
</script>
```

Otra forma de integrar el código de JavaScript es incrustar un fichero externo que contenga el código de JavaScript. Ésta sería la forma más recomendable, ya que así se consigue una separación entre el código y la estructura de la página web y como ventajas adicionales podrás compartir código entre diferentes páginas, centralizar el código para la depuración de errores, tendrás mayor claridad en tus desarrollos, más modularidad, seguridad del código y conseguirás que las páginas carguen más rápido. La rapidez de carga de las páginas se consigue al tener el código de JavaScript en un fichero independiente, ya que si más de una página tiene que acceder a ese fichero lo cogerá automáticamente de la caché del navegador con lo que se acelerará la carga de la página.

Para ello tendremos que añadir a la etiqueta script el atributo `src`, con el nombre del fichero que contiene el código de JavaScript. Generalmente los ficheros que contienen texto de JavaScript tendrán la extensión `.js`.

Por ejemplo:

```
<script  type="text/javascript"  src="tucodigo.js"></script>
```

Si necesitas cargar más de un fichero `.js` repite la misma instrucción cambiando el nombre del fichero. Las etiquetas de `<script>` y `</script>` son obligatorias a la hora de incluir el fichero `.js`. **Atención**: no escribas ningún código deJavaScript entre esas etiquetas cuando uses el atributo `src` .

Para **referenciar el fichero origen .js** de JavaScript dependerá de la localización física de ese fichero. Por ejemplo en la línea anterior el fichero `tucodigo.js` deberá estar en el mismo directorio que el fichero `.html`. Podrás enlazar fácilmente a otros ficheros de JavaScript localizados en directorios diferentes de tu servidor o de tu dominio. Si quieres hacer una referencia absoluta al fichero, la ruta tendrá que comenzar por `http://`, en otro caso tendrás que poner la ruta relativa dónde se encuentra tu fichero `.js`

Ejemplos:

```
<script  type="text/javascript"  src="http://www.tudominio.com/ejemplo.js"></script>
<script  type="text/javascript"  src="../js/ejemplo.js"></script>
```

(el fichero `ejemplo.js` se encuentra en el directorio anterior (`../`) al actual, dentro de la carpeta `js/`)

Cuando alguien examine el código fuente de tu página web verá el enlace a tu fichero `.js`, en lugar de ver el código de JavaScript directamente. Esto no quiere decir que tu código no sea inaccesible, ya que simplemente copiando la ruta de tu fichero `.js` y tecleándola en el navegador podremos descargar el fichero `.js` y ver todo el código de JavaScript. En otras palabras, nada de lo que tu navegador descargue para mostrar la página web podrá estar oculto de la vista de cualquier programador.

A veces te puedes encontrar que tu script se va a ejecutar en un navegador que no soporta JavaScript. Para ello dispones de una etiqueta `<noscript>`Texto informativo `</noscript>` que te permitirá indicar un texto adicional que se mostrará indicando que ese navegador no soporta JavaScript.

Si estamos trabajando con **XHTML**, la sintaxis para insertar un código de **JavaScript** directamente en el XHTML es un poco diferente:

```
<script  type="text/javascript">
   <!--//--><![CDATA[//><!--
      //  código de JavaScript a continuación
   //--><!]]>
</script>
```

Los analizadores de XHTML son intolerantes a los elementos que no comienzan por `<` y a las entidades HTML que no comienzan por `&`, y estos caracteres como verás más adelante son ampliamente utilizados en JavaScript. La solución es encapsular las instrucciones de JavaScript en lo que se conoce como sección `CDATA`:

```
<![CDATA[
     XHTML  permite cualquier tipo de carácter aquí incluyendo  <  y el símbolo &
]]>
```

Como puedes observar el código es un poco complejo en este caso, razón de más para integrar el código de JavaScript en un fichero externo.

Seguramente estarás pensando en cómo puedes **proteger el código de JavaScript** que vas a programar, del uso fraudulento por otros programadores o visitantes a tu página: la respuesta rápida a esa pregunta es que es imposible hacerlo.

Para que el código de JavaScript pueda ejecutarse correctamente deberá ser cargado por el navegador web y por lo tanto su código fuente deberá estar visible al navegador. Si realmente te preocupa que otras personas usen o roben tus scripts, deberías incluir un mensaje de copyright en tu código fuente. Piensa que no solamente tus scripts son visibles al mundo, sino que los scripts del resto de programadores también lo son. De esta forma puedes ver fácilmente cuando alguien está utilizando al pie de la letra tus scripts, aunque esto no evita que alguien copie tu código y elimine tu mensaje de copyright.

Lo más que se puede hacer es ofuscar el código, o hacerlo más difícil de entender. Las técnicas de ofuscación incluyen la eliminación de saltos de línea, espacios en blanco innecesarios, tabuladores, utilización de nombres ininteligibles en las funciones y variables, utilización de variables para almacenar trozos de código, uso de recursividad, etc. La forma más rápida de hacer todas esas tareas de ofuscación es utilizar un software que producirá una copia "comprimida" (*Reducir el espacio físico que ocupa el código fuente de nuestra aplicación, sin que se produzca pérdida de información*) del script que has programado para facilitar su carga rápida.

Para que veas un ejemplo de ofuscador de JavaScript visita:
http://www.javascriptobfuscator.com/
Lo mejor es que cambies ese paradigma y pienses de una manera diferente. En lugar de proteger tu código, lo mejor es promocionarlo y hacer ostentación de él, añadiendo comentarios útiles en el código fuente, publicándolo en tu blog o en webs de programación, etc. Incluso podrás añadir una

licencia **Creative Commons** para animar a la gente a que lo utilice, lo copie y lo mantenga público al resto del mundo. Así conseguirás mayor reputación como buen programador y la gente contactará contigo para más información, posibles trabajos, etc.

¿Te gustaría saber qué tipos de licencias tienes bajo Creative Commons?
http://es.creativecommons.org/licencia/

TEMA 2

Contenido

Estructura del lenguaje javascript

CASO PRÁCTICO
*En "BK Programación", han decidido ponerse manos a la obra y **Antonio** comienza a estudiar los **fundamentos del lenguaje JavaScript**, (siempre bajo la tutoría de **Juan**, que estará ahí para ayudarle en todo momento).*
En la unidad anterior Antonio analizó las posibilidades de los lenguajes de script, decidió que lenguaje emplearía para la programación en el entorno cliente y vio cómo insertar en la páginaHTML dicho lenguaje de script.
Cómo han decidido emplear el lenguaje Javascript, lo primero que hará Antonio es ver los fundamentos de dicho lenguaje, y cuál es la estructura básica para comenzar a programar lo más pronto posible.
***Ada** está muy entusiasmada con este proyecto y está convencida de que Antonio será capaz de hacer ese trabajo sin ningún problema.*

1.- Fundamentos de javascript.

Caso práctico
***Juan habla con Antonio** y le recomienda leer las bases y fundamentos de JavaScript, en los cuáles se explica cómo se ponen comentarios en el código, cómo se definen variables, estructuras de control, etc.*
Éste va a ser uno de los primeros pasos que Antonio tendrá que dar para el estudio de cualquier lenguaje de programación, y en especial de JavaScript.

El lenguaje que vas a estudiar ahora se llama JavaScript, pero quizás habrás oído otros nombres que te resulten similares como JScript (que es el nombre que le dio Microsoft a este lenguaje).
Microsoft le dio el nombre de JScript para evitar problemas relacionados con la marca, pero no pudo evitar otros problemas surgidos por las incompatibilidades que su versión de JavaScript tenía con múltiples navegadores. Para evitar esas incompatibilidades, el W3C, diseñó el DOM (Modelo de Objetos del Documento), que incorporaron las versiones de Internet Explorer 6, Netscape Navigator, Opera 7 y Mozilla Firefox desde su primera versión.

A partir de 1997 este lenguaje se rige por un estándar denominado ECMA, que se encarga de gestionar las especificaciones de este lenguaje de script (da igual el nombre que reciba). En el documentoECMA-262 es dónde se detallan dichas especificaciones. Tanto JavaScript como JScript son compatibles con el estándar ECMA-262.
JavaScript se diseñó con una sintaxis similar al lenguaje C y aunque adopta nombres y convenciones del lenguaje Java, éste último no tiene relación con JavaScript ya que tienen semánticas y propósitos diferentes.

JavaScript fue desarrollado originariamente por Brendan Eich, de Netscape, con el nombre de Mocha, el cual se renombró posteriormente a LiveScript y quedó finalmente como JavaScript.

Hoy en día JavaScript es una marca registrada de Oracle Corporation, y es usado con licencia por los productos creados por Netscape Communications y entidades actuales, como la fundaciónMozilla.

Más información sobre Brendan Eich. http://es.wikipedia.org/wiki/Brendan_Eich

Quién inventó JavaScript:

- ○ Microsoft.
- ○ Oracle Corporation.
- ◉ **Brendan Eich.**

El nombre original fue Mocha, luego LiveScript, para pasar a ser finalmente JavaScript.

1.1.- Comentarios en el código.

A la hora de programar en cualquier lenguaje de programación, es muy importante que comentes tu código.

Los comentarios son sentencias que el intérprete de JavaScript ignora. Sin embargo estas sentencias permiten a los desarrolladores dejar notas sobre cómo funcionan las cosas en sus scripts.

Los comentarios ocupan espacio dentro de tu código de JavaScript, por lo que cuando alguien se descargue vuestro código necesitará más o menos tiempo, dependiendo del tamaño de vuestro fichero. Aunque ésto pueda ser un problema, es muy recomendable el que documentes tu código lo máximo posible, ya que esto te proporcionará muchas más ventajas que inconvenientes.

JavaScript permite dos estilos de comentarios. Un estilo consiste en dos barras inclinadas hacia la derecha (sin espacios entre ellas), y es muy útil para comentar una línea sencilla. JavaScript ignorará cualquier carácter a la derecha de esas barras inclinadas en la misma línea, incluso si aparecen en el medio de una línea.

Ejemplos de comentarios de una única línea:

```
// Este es un comentario de una línea
var nombre="Marta" // Otro comentario sobre esta línea
// Podemos dejar por ejemplo
//
// una línea en medio en blanco
```

Para comentarios más largos, por ejemplo de una sección del documento, podemos emplear en lugar de las dos barras inclinadas el `/*` para comenzar la sección de comentarios, y `*/` para cerrar la sección de comentarios.

Por ejemplo:

```
/* Ésta es una sección
de comentarios
en el código de JavaScript */
```

O también:

```
/* ----------------------------------
función imprimir()
Imprime el listado de alumnos en orden alfabético
------------------------------------*/
function imprimir()
{
     // Líneas de código JavaScript
}
```

¿Sabías que usando comentarios podemos desactivar un bloque de código para que JavaScript deje de ejecutarlo sin tener que borrar nada en el código?
¿Sabías además, que los comentarios en JavaScript son muy útiles para dejar por ejemplo tu información de contacto para que otros programadores contacten contigo?

Más información y ejemplos sobre comentarios en JavaScript.
http://www.htmlpoint.com/javascript/corso/js_07.htm

1.2.- Variables.

La forma más conveniente de trabajar con datos en un script, es asignando primeramente los datos a una variable. Es incluso más fácil pensar que una variable es un cajón que almacena información. El tiempo que dicha información permanecerá almacenada, dependerá de muchos factores. En el momento que el navegador limpia la ventana o marco, cualquier variable conocida será eliminada.

Dispones de dos maneras de crear variables en JavaScript: una forma es usar la palabra reservada **var** seguida del nombre de la variable. Por ejemplo, para declarar una variable edad, el código de JavaScript será:

```
var edad;
```

Otra forma consiste en crear la variable, y asignarle un valor directamente durante la creación:

```
var edad = 38;
```

o bien, podríamos hacer:

```
var edad;
edad = 38; // Ya que no estamos obligados a declarar la variable pero es una buena práctica el hacerlo.
var altura, peso, edad: // Para declarar más de una variable en la misma línea.
```

La palabra reservada `var` se usa para la declaración o inicialización de la variable en el documento.
Una variable de JavaScript podrá almacenar diferentes tipos de valores, y una de las ventajas que tenemos con JavaScript es que no tendremos que decirle de qué tipo es una variable u otra.

A la hora de dar nombres a las variables, tendremos que poner nombres que realmente describan el contenido de la variable. No podremos usar palabras reservadas, ni símbolos de puntuación en el medio de la variable, ni la variable podrá contener espacios en blanco. Los nombres de las variables han de construirse con caracteres alfanuméricos y el carácter subrayado (_). No podremos utilizar caracteres raros como el signo `+`, `un espacio`, `%` , `$`, etc. en los nombres de variables, y estos nombres no podrán comenzar con un número.

Si queremos nombrar variables con dos palabras, tendremos que separarlas con el símbolo "_" o bien diferenciando las palabras con una mayúscula, por ejemplo:

```
var mi_peso;
var miPeso;  // Esta opción es más recomendable, ya que es más cómoda de escribir.
```

Deberemos tener cuidado también en no utilizar nombres reservados como variables. Por ejemplo, no podremos llamar a nuestra variable con el nombre de `return` o `for`.

Más información y ejemplos de Variables. **http://www.w3schools.com/js/js_variables.asp**

"La conciencia es la más variable de todas las reglas."
de VAUVENARGUES, Luc de Clapiers.

1.3.- Tipos de datos.

Las variables en JavaScript podrán contener cualquier tipo de dato. A continuación, se muestran los tipos de datos soportados en JavaScript:

Tipos de datos soportados por JavaScript		
Tipo	**Ejemplo**	**Descripción**
Cadena.	"Hola mundo"	Una serie de caracteres dentro de comillas dobles.
Número.	9.45	Un número sin comillas dobles.
Boolean.	true.	Un valor verdadero o falso.
Null.	null.	Desprovisto de contenido, simplemente es un valor null.
Object.		Es un objeto software que se define por sus propiedades y métodos (los arrays también son objetos).
Function.		La definición de una función.

El contener solamente este tipo de datos, simplifica mucho las tareas de programación, especialmente aquellas que abarcan tipos incompatibles entre números.

"Duda siempre de ti mismo, hasta que los datos no dejen lugar a dudas."
de BACH, Richard.

A continuación te presentamos un vídeo sobre tipos de datos en JavaScript.

http://www.youtube.com/watch?feature=player_embedded&v=cw5sFqlDBtE#!

Resumen: Vamos a ver dos artículos uno de ellos más genérico y otro más específico sobre los tipos de datos. Abrimos el editor web Komodo y creamos un directorio, en el cuál crearemos un archivo `ejemplos-tipos-datos.html`. Abrimos el código de JavaScript con `<script lenguaje="javascript">` y `</script>` . Primero hablaremos de los tipos de datos numéricos, los cuales no se diferencian en JavaScript. Es indiferente que se usen números reales o números enteros. Para JavaScript siempre serán tipos de datos numéricos. Creamos 4 variables que almacenan diferentes ejemplos de números. Para crear variables numéricas también se puede utilizar notación científica. Para ello usamos un exponente (que se define con una letra "e"), seguido a continuación del exponente al cual está elevado. También podremos crear números en bases diferentes (base 8, base 16). Para crear un número en base 8 comenzamos con un 0 seguido de números del 0 al 7. Para los números hexadecimal se pone un 0 seguido de una "x" y cualquier carácter hexadecimal (del 0 al 9 y de la A a la F).

Para crear variables de tipo cadena, simplemente tenemos que poner el texto entre comillas. Si tenemos una cadena que sólo contiene caracteres numéricos, para JavaScript eso será una cadena, independientemente del contenido que tenga. Dentro de las cadenas podemos usar caracteres de escape (como saltos de línea con `\n`, comillas dobles `\"`, tabuladores `\t`, etc.). Cuando sumamos cadenas en JavaScript lo que hacemos es concatenar esas cadenas (poner una a continuación de la otra).

Los tipos de datos booleano, son un `true` o un `false`. Y se usan mucho para tomar decisiones. Los únicos valores que podemos poner en las variables booleanas son `true` y `false`. Se muestra un ejemplo de uso de una variable booleana dentro de una sentencia `if`, modificando la comparación usando `==` y `===`.

Existen otros tipos de datos especiales que son de tipo Objeto, por ejemplo los **`arrays`**. Un array es una variable con un montón de casillas a las que se accede a través de un índice.

El programa final quedará así:

```
<script lenguaje="javascript">
   // tipo de datos numéricos
   // pueden ser enteros o números con decimales
   var miEntero = 33;
   var miDecimales = 2.5;
   var comaFlotante = 2344.983338;
   var numeral = 0.573;
   // pueden tener notación científica
   var numCientifico = 2.9e3;
   var otroNumCientifico = 2e-3;
   //alert(otroNumCientifico);
   // podemos escribir números en otras bases
   var numBase10 = 2200;
   var numBase8 = 0234;
   var numBase16 = 0x2A9F;
   //alert(numBase16);
   // tipo de datos cadena de caracteres
   var miCadena = "Hola! esto es una cadena!";
   var otraCadena = "2323232323"; // parece un numérico pero es una cadena
   // caracteres de escape en cadenas
   var cadenaConSaltoDeLinea = "linea1\nLínea2\nLínea3";
   var cadenaConComillas = "cadena 'con comillas' \"comillas dobles\"";
   var cadenaNum = "11";
   var sumaCadenaConcatenacion = otraCadena + cadenaNum;
   //alert(sumaCadenaConcatenacion);
   // tipos de datos booleano
```

```
   var miBooleano = true;
   var falso = false;
   if (niBooleano){
      //alert ("era true");
   }else{
      //alert("era false");
   }
   var booleano = (23==="23");
   //alerts(booleano)

   //esos eran los tipos de datos principales
   //pero existen otros tipos especiales que veremos más adelante
   //arrays
   var miArray = (2,5,7);
   //objetos
   var miObjeto = {
      propiedad: 23,
      otracosa: "hola"
   }

   //operador typeof para conocer un tipo
   alert("El tipo de miEntero es: " + typeof(miEntero));
   alert("El tipo de miCadena es: " + typeof(miCadena));
   alert("El tipo de miEntero es: " + typeof(miBooleano));
   alert("El tipo de miEntero es: " + typeof(miArray));
   alert("El tipo de miEntero es: " + typeof(miObjeto));
</script>
```

1.3.1.- Conversiones de tipos de datos.

Aunque los tipos de datos en JavaScript son muy sencillos, a veces te podrás encontrar con casos en los que las operaciones no se realizan correctamente, y eso es debido a la conversión de tipos de datos. JavaScript intenta realizar la mejor conversión cuando realiza esas operaciones, pero a veces no es el tipo de conversión que a ti te interesaría.

Por ejemplo cuando intentamos sumar dos números:

```
4 + 5 // resultado = 9
```

Si uno de esos números está en formato de cadena de texto, JavaScript lo que hará es intentar convertir el otro número a una cadena y los concatenará, por ejemplo:

```
4 + "5"       // resultado = "45"
```

Otro ejemplo podría ser:

```
4 + 5 + "6"   // resultado = "96"
```

Esto puede resultar ilógico pero sí que tiene su lógica. La expresión se evalúa de izquierda a derecha. La primera operación funciona correctamente devolviendo el valor de 9 pero al intentar sumarle una cadena de texto "6" JavaScript lo que hace es convertir ese número a una cadena de texto y se lo concatenará al comienzo del "6".

Para convertir cadenas a números dispones de las funciones: `parseInt()` y `parseFloat()`.

Por ejemplo:

```
parseInt("34")          // resultado = 34
parseInt("89.76")          // resultado = 89
```

parseFloat devolverá un entero o un número real según el caso:

```
parseFloat("34")          // resultado = 34
parseFloat("89.76")     // resultado = 89.76

4 + 5 + parseInt("6")        // resultado = 15
```

Si lo que deseas es realizar la conversión de números a cadenas, es mucho más sencillo, ya que simplemente tendrás que concatenar una cadena vacía al principio, y de esta forma el número será convertido a su cadena equivalente:

```
("" + 3400)          // resultado = "3400"
("" + 3400).length     // resultado = 4
```

En el segundo ejemplo podemos ver la gran potencia de la evaluación de expresiones. Los paréntesis fuerzan la conversión del número a una cadena. Una cadena de texto en JavaScript tiene una propiedad asociada con ella que es la longitud (length), la cuál te devolverá en este caso el número 4, indicando que hay 4 caracteres en esa cadena "3400". La longitud de una cadena es un número, no una cadena.

1.4.- Operadores.

JavaScript es un lenguaje rico en operadores: símbolos y palabras que realizan operaciones sobre uno o varios valores, para obtener un nuevo valor.

Cualquier valor sobre el cual se realiza una acción (indicada por el operador), se denomina un operando. Una **expresión** puede contener un operando y un operador (denominado operador unario), como por ejemplo en `b++`, o bien dos operandos, separados por un operador (denominado operador binario), como por ejemplo en `a + b`.

Categorías de operadores en JavaScript	
Tipo	**Qué realizan**
Comparación.	Comparan los valores de 2 operandos, devolviendo un resultado de true o false (se usan extensivamente en sentencias condicionales como `if...` `else` y en instrucciones `loop`). `==  !=  === !==  >  >=  <  <=`
Aritméticos.	Unen dos operandos para producir un único valor que es el resultado de una operación aritmética u otra operación sobre ambos operandos. `+  -  *  /  %  ++  - -  +valor  -valor`
Asignación.	Asigna el valor a la derecha de la expresión a la variable que está a la izquierda. `=  +=  -=  *=  /=  %=  <<=  >=  >>=  >>>=  &=  |=  ^=  []`
Boolean.	Realizan operaciones booleanas aritméticas sobre uno o dos operandos booleanos. `&&  ||  !`
Bit a Bit.	Realizan operaciones aritméticas o de desplazamiento de columna en las representaciones binarias de dos operandos. `&	^ ~ << >> >>>`
Objeto.	Ayudan a los scripts a evaluar la herencia y capacidades de un objeto particular antes de que tengamos que invocar al objeto y sus propiedades o métodos. `.  []  ()  delete  in  instanceOf  new  this`
Misceláneos.	Operadores que tienen un comportamiento especial. `,  ?:  typeof  void`

Operadores. **http://www.htmlpoint.com/javascript/corso/js_30.htm**

1.4.1.- Operadores de comparación.

Operadores de comparación en JavaScript

Operadores de comparación en JavaScript			
Sintaxis	**Nombre**	**Tipos de operandos**	**Resultados**
==	Igualdad.	Todos.	Boolean.
!=	Distinto.	Todos.	Boolean.

===	Igualdad estricta.	Todos.	Boolean.
!==	Desigualdad estricta.	Todos.	Boolean.
>	Mayor que .	Todos.	Boolean.
>=	Mayor o igual que.	Todos.	Boolean.
<	Menor que.	Todos.	Boolean.
<=	Menor o igual que.	Todos.	Boolean.

En valores numéricos, los resultados serían los mismos que obtendríamos con cálculos algebraicos. Por ejemplo:

```
30  ==  30      //  true
30  ==  30.0    //  true
5  !=  8        //  true
9  >  13        //  false
7.29  <=  7.28   //  false
```

También podríamos comparar cadenas a este nivel:

```
"Marta"  ==  "Marta"       //  true
"Marta"  ==  "marta"       //  false
"Marta"  >  "marta"      //  false
"Mark"  <  "Marta"      //  true
```

Para poder comparar cadenas, JavaScript convierte cada carácter de la cadena de un string, en su correspondiente valor ASCII. Cada letra, comenzando con la primera del operando de la izquierda, se compara con su correspondiente letra en el operando de la derecha. Los valores ASCII de las letras mayúsculas, son más pequeños que sus correspondientes en minúscula, por esa razón "Marta" no es mayor que "marta". En JavaScript hay que tener muy en cuenta la sensibilidad a mayúsculas y minúsculas.

Si por ejemplo comparamos un número con su cadena correspondiente:

```
"123" == 123          // true
```

JavaScript cuando realiza esta comparación, convierte la cadena en su número correspondiente y luego realiza la comparación. También dispones de otra opción, que consiste en convertir mediante las funciones `parseInt()` o `parseFloat()` el operando correspondiente:

```
parseInt("123") == 123          // true
```

Los operadores `===` y `!==` comparan tanto el dato como el tipo de dato. El operador `===` sólo devolverá `true`, cuando los dos operandos son del mismo tipo de datos (por ejemplo ambos son números) y tienen el mismo valor.

Operadores de Comparación. **http://www.webestilo.com/javascript/js06.phtml**

1.4.2.- Operadores aritméticos.

Operadores ariméticos en JavaScript

Operadores aritméticos en JavaScript			
Sintaxis	**Nombre**	**Tipos de Operando**	**Resultados**
+	Más.	integer, float, string.	integer, float, string.
-	Menos.	integer, float.	integer, float.
*	Multiplicación.	integer, float.	integer, float.
/	División.	integer, float.	integer, float.
%	Módulo.	integer, float.	integer, float.
++	Incremento.	integer, float.	integer, float.

--	Decremento.	integer, float.	integer, float.
+valor	Positivo.	integer, float, string.	integer, float.
-valor	Negativo.	integer, float, string.	integer, float.

Veamos algunos ejemplos:

```
var a = 10;        // Inicializamos a al valor 10
var z = 0;      // Inicializamos z al valor 0
z = a;          // a es igual a 10, por lo tanto z es igual a 10.
z = ++a;      // el valor de a se incrementa justo antes de ser asignado a z, por lo que a es 11 y z valdrá 11.
z = a++;      // se asigna el valor de a (11) a z y luego se incrementa el valor de a (pasa a ser 12).
z = a++;       // a vale 12 antes de la asignación, por lo que z es igual a 12; una vez hecha la asignación a valdrá 13.
```

Otros ejemplos:

```
var x = 2;
var y = 8;
var z = -x;      // z es igual a -2, pero x sigue siendo igual a 2.
z = - (x + y);      // z es igual a -10, x es igual a 2 e y es igual a 8.
z = -x + y;      // z es igual a 6, pero x sigue siendo igual a 2 e y igual a 8.
```

"La música es la aritmética de los sonidos, como la óptica es la geometría de la luz."

de DEBUSSY, Claude.

Si sumamos 9+4+"10" en JavaScript obtendremos:

- ○ "23".
- ○ 23.
- ◉ **"1310".**

Primero se sumarán 9+4 y a ese resultado 13, se le concatenará la cadena "10".

1.4.3.- Operadores de asignación.

Operadores de asignación en JavaScript.

Operadores de asignación en JavaScript			
Sintaxis	**Nombre**	**Ejemplo**	**Significado**
=	Asignación.	x = y	x = y
+=	Sumar un valor.	x += y	x = x + y
-=	Substraer un valor.	x -= y	x = x - y
***=**	Multiplicar un valor.	x *= y	x = x * y
/=	Dividir un valor.	x /= y	x = x / y
%=	Módulo de un valor.	x %= y	x = x % y
<<=	Desplazar bits a la izquierda.	x <<= y	x = x << y
>=	Desplazar bits a la derecha.	x >= y	x = x > y
>>=	Desplazar bits a la derecha rellenando con 0.	x >>= y	x = x >> y
>>>=	Desplazar bits a la derecha.	x >>>= y	x = x >>> y
&=	Operación AND bit a bit.	x &= y	x = x & y
|=	Operación OR bit a bit.	x |= y	x = x | y
^=	Operación XOR bit a bit.	x ^ = y	x = x ^ y
[]=	Desestructurando asignaciones.	[a,b]=[c,d]	a=c, b=d

Operadores. **http://www.w3schools.com/js/js_operators.asp**

"Los esfuerzos, cuando se suman, se multiplican."

Anónimo.

1.4.4.- Operadores booleanos.

Debido a que parte de la programación tiene un gran componente de lógica, es por ello, que los operadores booleanos juegan un gran papel.

Los operadores booleanos te van a permitir evaluar expresiones, devolviendo como resultado `true` (verdadero) o `false` (falso).

Operadores de boolean en JavaScript			
Sintaxis	**Nombre**	**Operandos**	**Resultados**
&&	AND.	Boolean.	Boolean.
||	OR.	Boolean.	Boolean.
!	Not.	Boolean.	Boolean.

Ejemplos:

```
!true            //  resultado =  false
!(10  >  5)             //  resultado  =  false
!(10  <  5)             //  resultado  =  true
!("gato"  ==  "pato")        //  resultado  =  true

5  >  1  &&  50  >  10        //  resultado  =  true
5  >  1  &&  50  <  10        //  resultado  =  false
5  <  1  &&  50  >  10        //  resultado  =  false
5  <  1  &&  50  <  10        //  resultado  =  false
```

Tabla de valores de verdad del operador AND			
Operando Izquierdo	**Operador AND**	**Operando Derecho**	**Resultado**
True	&&	True	True
True	&&	False	False
False	&&	True	False
False	&&	False	False

Tabla de valores de verdad del operador OR			
Operando Izquierdo	**Operador OR**	**Operando Derecho**	**Resultado**
True	||	True	True
True	||	False	True
False	||	True	True
False	||	False	False

Ejemplos:

```
5  >  1  ||  50  >  10        //  resultado  =  true
5  >  1  ||  50  <  10        //  resultado  =  true
5  <  1  ||  50  >  10        //  resultado  =  true
5  <  1  ||  50  <  10        //  resultado  =  false
```

1.4.5.- Operadores bit a bit.

Para los programadores de scripts, las operaciones bit a bit suelen ser un tema avanzado. A menos que tú tengas que gestionar procesos externos en aplicaciones del lado del servidor, o la conexión con applets de Java, es raro que tengas que usar este tipo de operadores.

Los operandos numéricos, pueden aparecer en JavaScript en cualquiera de los tres formatos posibles (decimal, octal o hexadecimal). Tan pronto como el operador tenga un operando, su valor se convertirá a representación binaria (32 bits de longitud). Las tres primeras operaciones binarias bit a bit que podemos realizar son**AND**, **OR** y **XOR** y los resultados de comparar bit a bit serán:
Bit a bit AND: 1 si ambos dígitos son 1.
Bit a bit OR: 1 si cualquiera de los dos dígitos es 1.
Bit a bit XOR: 1 si sólo un dígito es 1.

Tabla de operador Bit a Bit en JavaScript			
Operador	**Nombre**	**Operando izquierdo**	**Operando derecho**
&	Desplazamiento AND.	Valor integer.	Valor integer.
|	Desplazamiento OR.	Valor integer.	Valor integer.
^	Desplazamiento XOR.	Valor integer.	Valor integer.
~	Desplazamiento NOT.	(Ninguno).	Valor integer.
<<	Desplazamiento a la izquierda.	Valor integer.	Cantidad a desplazar.
>>	Desplazamiento a la derecha.	Valor integer.	Cantidad a desplazar.
>>>	Desplazamiento derecha rellenando con 0.	Valor integer.	Cantidad a desplazar.

Por ejemplo:

```
4 << 2 // resultado = 16
```

La razón de este resultado es que el número decimal **4** en binario es `00000100`. El operador `<<` indica a JavaScript que desplace todos los dígitos dos lugares hacia la izquierda, dando como resultado en binario `00010000`, que convertido a decimal te dará el valor **16**.

En el siguiente enlace podrás ver ejemplos de operaciones a nivel de bits:
http://www.mundoprogramacion.com/net/dotnet/operar_con_bits.aspx

1.4.6.- Operadores de objeto.

El siguiente grupo de operadores se relaciona directamente con objetos y tipos de datos. La mayor parte de ellos fueron implementados a partir de las primeras versiones de JavaScript, por lo que puede haber algún tipo de incompatibilidad con navegadores antiguos.

. (punto)

El operador punto, indica que el objeto a su izquierda tiene o contiene el recurso a su derecha, como por ejemplo: `objeto.propiedad` y `objeto.método()`.
Ejemplo con un objeto nativo de JavaScript:

```
var s = new String('rafa');
var longitud = s.length;
var pos = s.indexOf("fa"); // resultado: pos = 2
```

[] (corchetes para enumerar miembros de un objeto).

Por ejemplo cuando creamos un array: `var a =["Santiago","Coruña", "Lugo"];`
Enumerar un elemento de un array: `a[1] = "Coruña";`
Enumerar una propiedad de un objeto: `a["color"]= "azul";`

Delete (para eliminar un elemento de una colección).

Por ejemplo si consideramos:

```
var oceanos = new Array("Atlantico", "Pacifico", "Indico","Artico");
```

Podríamos hacer:
```
delete oceanos[2];
```

Ésto eliminaría el tercer elemento del array ("Indico"), pero la longitud del array no cambiaría. Si intentamos referenciar esa posición oceanos[2] obtendríamos undefined.

In (para inspeccionar métodos o propiedades de un objeto).

El operando a la izquierda del operador, es una cadena referente a la propiedad o método (simplemente el nombre del método sin paréntesis); el operando a la derecha del operador, es el objeto que estamos inspeccionando. Si el objeto conoce la propiedad o método, la expresión devolverá `true`.

Ejemplo: `"write" in document`
o también `"defaultView" in document`

instanceof (para comprobar si un objeto es una instancia de un objeto nativo de JavaScript).

Ejemplo:
```
a = new Array(1,2,3);
a instanceof Array; // devolverá true.
new (para acceder a los constructores de objetos incorporados en el núcleo de JavaScript).
```

Ejemplo:
```
var hoy = new Date();
// creará el objeto hoy de tipo Date() empleando el constructor por defecto de dicho objeto.
this (para hacer referencia al propio objeto en el que estamos localizados).
```

Ejemplo:
```
nombre.onchange = validateInput;
function  validateInput(evt)
          {
               var  valorDeInput  =  this.value;
               // Este this hace referencia al objeto nombre que estamos validando.
          }
```

1.4.7.- Operadores misceláneos.

El operador coma ,

Este operador, indica una serie de expresiones que van a ser evaluadas en secuencia, de izquierda a derecha. La mayor parte de las veces, este operador se usa para combinar múltiples declaraciones e inicializaciones de variables en una única línea.
Ejemplo:
```
var nombre, direccion, apellidos, edad;
```

Otra situación en la que podemos usar este operador coma, es dentro de la expresión `loop`. En el siguiente ejemplo inicializamos dos variables de tipo contador, y las incrementamos en diferentes porcentajes. Cuando comienza el bucle, ambas variables se inicializan a 0 y a cada paso del bucle una de ellas se incrementa en 1, mientras que la otra se incrementa en 10.
```
for  (var  i=0,  j=0 ;  i  <  125;  i++,  j+10)
{
     // más instrucciones aquí dentro
}
```
Nota: no confundir la coma `,` con el delimitador de parámetros "`;`" en la instrucción `for`.

? : (operador condicional)

Este operador condicional es la forma reducida de la expresión `if …. else`.

La sintaxis formal para este operador condicional es:

```
condicion ? expresión si se cumple la condición: expresión si no se cumple;
```

Si usamos esta expresión con un operador de asignación:

```
var = condicion ? expresión si se cumple la condición: expresión si no se cumple;
```

Ejemplo:

```
     var a,b;
     a = 3; b = 5;
     var h = a > b ? a : b;      // a h se le asignará el valor 5;
```

typeof (devuelve el tipo de valor de una variable o expresión).

Este operador unario se usa para identificar cuando una variable o expresión es de alguno de los siguientes tipos: `number`, `string`, `boolean`, `object`, `function` o `undefined`.

Ejemplo:

```
if  (typeof  miVariable  ==  "number")
{
     miVariable  =  parseInt(miVariable);
}
```

1.5.- Condiciones y bucles.

En esta sección te mostraremos cómo los programas pueden tomar decisiones, y cómo puedes lograr que un script repita un bloque de instrucciones las veces que quieras.
Cuando te levantas cada día tomas decisiones de alguna clase; muchas veces ni te das cuenta de ello, pero lo estás haciendo. Por ejemplo, imagínate que vas a hacer la compra a un supermercado; desde el momento que entras en el supermercado ya estás tomando decisiones: ¿compro primero la leche o compro la verdura?, ¿ese precio es barato o es caro?, ¿el color de ese tinte es azul claro u oscuro?, ¿tengo suficiente dinero para pagar o no?, ¿me llegan estos kilogramos de patatas o no?, ¿pago en efectivo o tarjeta?, etc.
Es decir, tomamos innumerables decisiones a lo largo del día y la mayor parte de las veces no nos damos ni cuenta de ello.

En las siguientes secciones, verás cómo puedes ejecutar unas u otras instrucciones, dependiendo de ciertas condiciones, y cómo puedes repetir una o varias instrucciones, las veces que te hagan falta.

Es correcta la instrucción if (a=b) then suma=a+b;

○ Sí ◉ No

Muy bien, ya que en la condición estamos haciendo una asignación. Tendríamos que escribir == (dos iguales seguidos), para que la condición pudiera ser evaluada.

"Imponer condiciones excesivamente duras es dispensar de su cumplimiento."

Anónimo.

1.5.1.- Estructuras de control.

En los lenguajes de programación, las instrucciones que te permiten controlar las decisiones y bucles de ejecución, se denominan "Estructuras de Control". Una estructura de control, dirige el flujo de ejecución a través de una secuencia de instrucciones, basadas en decisiones simples y en otros factores.

Una parte muy importante de una estructura de control es la "condición". Cada condición es una expresión que se evalúa a `true` o `false`.

En JavaScript tenemos varias estructuras de control, para las diferentes situaciones que te puedas encontrar durante la programación. Tres de las estructuras de control más comunes son: construcciones **if**, construcciones **`if...else`** y los **`bucles`**.

Construcción if

La decisión más simple que podemos tomar en un programa, es la de seguir una rama determinada si una determinada condición es `true` .

Sintaxis:

```
if  (condición)     // entre paréntesis irá la condición que se evaluará a true o false.
{
     // instrucciones a ejecutar si se cumple la condición
}
```

Ejemplo:

```
if (miEdad >30)
{
     alert("Ya eres una persona adulta");
}
```

Construcción if … else

En este tipo de construcción, podemos gestionar que haremos cuando se cumpla y cuando no se cumpla una determinada condición.

Sintaxis:

```
if  (condición)     // entre paréntesis irá la condición que se evaluará a true o false.
{
     // instrucciones a ejecutar si se cumple la condición
}
else
{
     // instrucciones a ejecutar si no se cumple la condición
}
```

Ejemplo:

```
if (miEdad >30)
{
     alert("Ya eres una persona adulta.");
}
else
{
     alert("Eres una persona joven.");
}
```

Ejemplos de if..else. **http://www.gratismil.com/apuntes/javascript/21-if-else-en-javascript.htm**

1.5.2.- Bucles.

Los bucles son estructuras repetitivas, que se ejecutarán un número de veces fijado expresamente, o que dependerá de si se cumple una determinada condición.

Bucle for.

Este tipo de bucle te deja repetir un bloque de instrucciones un número limitado de veces.

Sintaxis:

```
for (expresión inicial; condición; incremento)
{
     // Instrucciones a ejecutar dentro del bucle.
```

```
}
```

Ejemplo:

```
for  (var  i=1;  i<=20;  i++)
{
     // instrucciones que se ejecutarán 20 veces.
}
```

Bucle while().

Este tipo de bucles se utilizan cuando queremos repetir la ejecución de unas sentencias un número indefinido de veces, siempre que se cumpla una condición. Es más sencillo de comprender que el bucle **FOR**, ya que no incorpora en la misma línea la inicialización de las variables, su condición para seguir ejecutándose y su actualización. Sólo se indica, como veremos a continuación, la condición que se tiene que cumplir para que se realice una iteración o repetición.

Sintaxis:

```
while (condición)
{
     // Instrucciones a ejecutar dentro del bucle.
}
```

Ejemplo:

```
var i=0;
while (i <=10)
{
     // Instrucciones a ejecutar dentro del bucle hasta que i sea mayor que 10 y no se cumpla
la condición.
     i++;
}
```

Bucle do … while().

El tipo de bucle **do...while** es la última de las estructuras para implementar repeticiones de las que dispone JavaScript, y es una variación del bucle **while()** visto anteriormente. Se utiliza generalmente, cuando no sabemos el número de veces que se habrá de ejecutar el bucle. Es prácticamente igual que el bucle **while()**, con la diferencia, de que sabemos seguro que el bucle por lo menos se ejecutará una vez.

Sintaxis:

```
do {
     // Instrucciones a ejecutar dentro del bucle.
}while (condición);
```

Ejemplo:

```
var a = 1;
do{
     alert("El valor de a es: "+a);     // Mostrará esta alerta 2 veces.
     a++;
}while (a<3);
```

1.5.3.- Ejemplo sencillo con JavaScript.

Aquí se muestra el código fuente, de un pequeño ejemplo de una aplicación en JavaScript.

Simplemente copia el código, pégalo en tu editor favorito y guarda el archivo con la extensión .html Para probar la aplicación haz doble click en el fichero .html y visualízalo con tu navegador.

```
<!DOCTYPE      html      PUBLIC      "-//W3C//DTD      XHTML      1.0      Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Ejemplo Básico de JavaScript</title>
```

```
</head>
<body>
<h2>Código fuente de un ejemplo básico con JavaScript
</h2>
<script type="text/javascript">
// Definimos variables.
var nombre, apellidos, edad, hermanos;

nombre="Rafa";
apellidos="Martinez Díaz";
edad=38;
hermanos=3;

// Imprimo el nombre y los apellidos.
document.write("Hola " + nombre + " " + apellidos);

// Imprimo un salto de línea.
document.write("<br/>");

// Imprime la edad y el número de hermanos.
document.write(nombre + " tienes " + edad + " años y además tienes " + hermanos + "
hermanos.<br/>");

// Fíjate en la diferencia entre las dos siguientes lineas.
document.write("Dentro de 15 años tu edad será " + edad + 15 + ".<br/>");
document.write("Dentro de 15 años tu edad será " + (edad+15) + ".<br/>");

// Tu nombre escrito 50 veces.
for (i=1; i<=50; i++)
{
     document.write(nombre + ",");
}
</script>
</body>
</html>
```

Información y ejemplos sobre JavaScript. **http://www.w3schools.com/js/default.asp**

TEMA 3

Contenido

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.*

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 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.

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.

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

- (•) Sí.
- ( ) No.
- ( ) Depende de si lleva o no parámetros.

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

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.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 cual nos encontramos.

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

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

```
<!DOCTYPE  html>
<html>
   <head>
      <meta  http-equiv="content-type"  content="text/html;charset=utf-8">
      <title>Apertura y Cierre de Ventanas</title>
      <script type="text/javascript">
         function  inicializar(){
            document.getElementById("crear-ventana").onclick=crearNueva;
            document.getElementById("cerrar-ventana").onclick=cerrarNueva;
```

```
        }

        var  nuevaVentana;
        function  crearNueva(){
           nuevaVentana = window.open("http://www.google.es","","height=400,width=800");
        }

        function  cerrarNueva(){
           if (nuevaVentana){
              nuevaVentana.close(); nuevaVentana  =  null;
           }
        }
     </script>
  </head>
  <body onLoad="inicializar()">
     <h1>Abrimos y cerramos ventanas</h1>
     <form>
        <p> <input  type="button"  id="crear-ventana"  value="Crear Nueva Ventana">
        <input  type="button"  id="cerrar-ventana"  value="Cerrar Nueva Ventana"> </p>
     </form>
  </body>
</html>
```

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

Métodos del objeto Window	
Método	**Descripción**
alert()	Muestra una ventana emergente de alerta y un botón de aceptar.
blur()	Elimina el foco de la ventana actual.
clearInterval()	Resetea el cronómetro ajustado con setInterval().
setInterval()	Llama a una función o evalúa una expresión en un intervalo especificado (en milisegundos).
close()	Cierra la ventana actual.
confirm()	Muestra una ventana emergente con un mensaje, un botón de aceptar y un botón de cancelar.
focus()	Coloca el foco en la ventana actual.

`open()`	Abre una nueva ventana de navegación.
`prompt()`	Muestra una ventana de diálogo para introducir datos.

En el siguiente anexo puedes ampliar la información sobre el objeto **Window** y todas sus propiedades y métodos.

Más información y ejemplos 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.

Propiedades del objeto Location	
Propiedad	**Descripción**
`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 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()`	Reemplaza el historial actual mientras carga la URL especificada en cadenaURL.

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

MONTAIGNE, Michel de.

El siguiente anexo 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**
`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.
`platform`	Cadena con la plataforma sobre la que se está ejecutando el programa cliente.
`userAgent`	Cadena que contiene la cabecera completa del agente enviada en una petición HTTP. Contiene la información de las propiedades `appCodeName` y `appVersion`.

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

El siguiente enlace amplía información sobre el objeto **Navigator** y todas sus propiedades y métodos.
Más información y ejemplos sobre el objeto Navigator.

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.**

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

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**
`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 del objeto Document	
Propiedad	**Descripción**
`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.

Propiedades del objeto Document	
Método	**Descripción**
`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
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**
`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`.
`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`.

Eventos del objeto Frame/Iframe	
Evento	**Descripción**
`onload`	Script que se ejecutará inmediatamente después a que se cargue el `frame`/`iframe`.

"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 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.

"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.

Ejemplo2.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>
```

Ejemplo2_1.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 con el ejemplo anterior, a lo mejor tienes que desactivar el bloqueador de pop-ups y volver a probar.

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

YANG, Jerry.

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.

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

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.

Esto 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, esto 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`.

"Si me hubieran hecho objeto sería objetivo, pero me hicieron sujeto."
BERGAMÍN, José.

¿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**
\"	Comillas dobles.
\'	Comilla simple.
\\	Barra inclinada.
\b	Retroceso.
\t	Tabulador.
\n	Nueva línea.
\r	Salto de línea.
\f	Avance de página.

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

Más información y ejemplos sobre el objeto String.

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**
`length`	Contiene la longitud de una cadena.

Métodos del objeto String	
Métodos	**Descripción**
`charAt()`	Devuelve el carácter especificado por la posición que se indica entre paréntesis.
`charCodeAt()`	Devuelve el Unicode del carácter especificado por la posición que se indica entre paréntesis.
`concat()`	Une una o más cadenas y devuelve el resultado de esa unión.

`fromCharCode()`	Convierte valores Unicode a caracteres.
`indexOf()`	Devuelve la posición de la primera ocurrencia del carácter buscado en la cadena.
`lastIndexOf()`	Devuelve la posición de la última ocurrencia del carácter buscado en la cadena.
`match()`	Busca una coincidencia entre una expresión regular y una cadena y devuelve las coincidencias o null si no ha encontrado nada.
`replace()`	Busca una subcadena en la cadena y la reemplaza por la nueva cadena especificada.
`search()`	Busca una subcadena en la cadena y devuelve la posición dónde se encontró.
`slice()`	Extrae una parte de la cadena y devuelve una nueva cadena.
`split()`	Divide una cadena en un array de subcadenas.
`substr()`	Extrae los caracteres de una cadena, comenzando en una determinada posición y con el número de caracteres indicado.
`substring()`	Extrae los caracteres de una cadena entre dos índices especificados.
`toLowerCase()`	Convierte una cadena en minúsculas.
`toUpperCase()`	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**
`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 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 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 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**
`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 del objeto Number	
Método	**Descripcción**
`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

	hexadecimal.
`valueOf()`	Devuelve el valor primitivo de un objeto `Number`.

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	
Popiedad	**Descripción**
`constructor`	Devuelve la función que creó el objeto `Boolean`.
`prototype`	Te permitirá añadir propiedades y métodos a un objeto.
Métodos del objeto Boolean	
Método	**Descripción**
`toString()`	Convierte un valor `Boolean` a una cadena y devuelve el resultado.
`valueOf()`	Devuelve el valor primitivo de un objeto `Boolean`.

Algunos ejemplos de uso:

```
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
```

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

- ○ Sí
- ◉ **No**

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

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 `Date`.
`prototype`	Te permitirá añadir propiedades y métodos a un objeto.

Métodos del objeto Date	
Método	**Descripción**
`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).

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);
```

ANEXO I - EL OBJETO WINDOW

Se trata del objeto **más alto en la jerarquía del navegador** (`navigator` es un objeto independiente de todos en la jerarquía), pues todos los componentes de una página web están situados dentro de una ventana. El objeto `window` hace referencia a la ventana actual. Veamos a continuación sus propiedades y sus métodos.

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.

Me dejo en el tintero otras propiedades y métodos como `innerHeight`, `innerWidth`, `outerHeight`, `outerWidth`, `pageXOffset`, `pageYOffset`, `personalbar`, `scrollbars`, `back()`, `find(["cadena"],[caso,bkwd])`, `forward()`, `home()`, `print()`, `stop()`... todas ellas disponibles a partir de NS 4 y cuya explicación remito como ejercicio al lector interesado en saber más sobre el objeto window.

```
<HTML>
   <HEAD>
      <title>Ejemplo de JavaScript</title>
      <script LANGUAGE="JavaScript">
         <!--
            function moverVentana(){
               mi_ventana.moveBy(5,5);
               i++;
               if (i<20)
                  setTimeout('moverVentana()',100);
               else
                  mi_ventana.close();
               }
         //-->
      </script>

   </HEAD>
   <BODY>
      <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();
         //-->
      </script>
   </BODY>
</HTML>
```

ANEXO II - EL OBJETO LOCATION

Este objeto contiene la URL actual así como algunos datos de interés respecto a esta URL. Su finalidad principal es, por una parte, modificar el objeto `location` para cambiar a una nueva URL, y extraer los componentes de dicha URL de forma separada para poder trabajar con ellos de forma individual si es el caso. Recordemos que la sintaxis de una URL era:
`protocolo://maquina_host[:puerto]/camino_al_recurso`

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.

```
<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>
```

ANEXO III - EL OBJETO NAVIGATOR

Este objeto simplemente nos da información relativa al navegador que esté utilizando el usuario.

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.

```
<HTML>
   <HEAD>
      <title>Ejemplo de JavaScript</title>
   </HEAD>
   <BODY>
      <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>
   </BODY>
</HTML>
```

ANEXO IV - EL OBJETO STRING

El objeto String se utiliza para manipular una cadena almacenada de texto.
Los objetos String se crean con new String()

Sintaxis

```
var txt = new String("cadena");
```

o simplemente:

```
var txt = "cadena";
```

Para obtener más información consulte:
http://www.w3schools.com/js/js_obj_string.asp

Propiedades del objeto String

Propiedad	Descripción
`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 del objeto String

Método	Descripción
`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

Métodos envolventes de String HTML

Los métodos envolventes HTML devuelven la cadena envuelta en una etiqueta HTML apropiada

Method	Description
`anchor()`	Crea un `anchor` (ancla)
`big()`	Visualiza una cadena usando una fuente grande
`blink()`	Visualiza una cadena parpadeante
`bold()`	Visualiza una cadena en negrita

`fixed()`	Visualiza una cadena utilizando una fuente de paso fijo
`fontcolor()`	Visualiza una cadena usando el color especificado
`fontsize()`	Visualiza una cadena usando el tamaño especificado
`italics()`	Visualiza una cadena en cursiva
`link()`	Visualiza una cadena como un vínculo
`small()`	Visualiza una cadena en letra pequeña
`strike()`	Visualiza una cadena con un tachado
`sub()`	Visualiza una cadena como texto subíndice
`sup()`	Visualiza una cadena como texto superíndice

TEMA 4

Contenido

Estructuras definidas por el usuario en JavaScript

CASO PRÁCTICO
Antonio *ha avanzado mucho en su estudio de JavaScript, viendo los principales objetos con los que puede trabajar, sus métodos y propiedades.*
Ha llegado el momento, de ver cómo puede organizar los datos que genera en su aplicación de JavaScript, de forma que le permita gestionarlos más eficientemente.
También verá cómo crear funciones, el alcance de las variables y cómo utilizarlas en su código. Y para terminar ***Juan****, le va a explicar un poco más en detalle, cómo podrá crear nuevos objetos y asignarles propiedades y métodos en JavaScript.*
Antonio *ha comenzado a analizar más a fondo parte del trabajo que tiene que realizar, y comenta con su directora* ***Ada****, la posibilidad de hacer algunos bocetos de las innovaciones y mejoras que quiere aportar al proyecto.* ***Ada*** *está muy contenta con los progresos de* ***Antonio*** *y está deseando ver los bocetos.*

1.- Estructuras de datos.

Caso práctico
Antonio *comienza a estudiar las estructuras de datos que le permitirán almacenar la información que gestionará en sus scripts, de una forma mucho más organizada. Existen muchos tipos de estructuras de datos, pero en este caso* ***Antonio*** *profundizará en una de ellas "los arrays".*
Verá cómo crearlos, recorrerlos para consultar su información, borrar elementos del array, y las propiedades y métodos disponibles para su gestión.
Terminará estudiando lo que son arrays paralelos y arrays multidimensionales, los cuales le permitirán ampliar la capacidad y ventajas que aporta un array estándar.

En los lenguajes de programación existen estructuras de datos especiales, que nos sirven para guardar información más compleja que una variable simple. En la segunda unidad vimos como crear variables y cómo almacenar valores (simples), dentro de esas variables.

Hay un montón de tipos de estructuras de datos (listas, pilas, colas, árboles, conjuntos,…), que se pueden utilizar para almacenar datos, pero una estructura de las más utilizadas en todos los lenguajes es el array. El **array**, es como una variable o zona de almacenamiento continuo, donde podemos introducir varios valores en lugar de solamente uno, como ocurre con las variables normales.

Los arrays también se suelen denominar **matrices** o **vectores**. Desde el punto de vista lógico, una matriz se puede ver como un conjunto de elementos ordenados en fila (o filas y columnas si tuviera dos o más dimensiones).

Se puede considerar que todos los arrays son de una dimensión: la dimensión principal, pero los elementos de dicha fila pueden a su vez contener otros arrays o matrices, lo que nos permitiría hablar de **arrays multidimensionales** (los más fáciles de imaginar son los de una, dos y tres dimensiones).

Los arrays son una estructura de datos, adecuada para situaciones en las que el acceso a los datos se realiza de forma aleatoria e impredecible. Por el contrario, si los elementos pueden estar ordenados y se va a utilizar acceso secuencial sería más adecuado usar una lista, ya que esta estructura puede cambiar de tamaño fácilmente durante la ejecución de un programa.

Los arrays nos permiten guardar un montón de elementos y acceder a ellos de manera independiente. Cada elemento es referenciado por la posición que ocupa dentro del array. Dichas posiciones se llaman **índices** y siempre son correlativos. Existen tres formas de indexar los elementos de un array:

- ✓ **indexación base-cero(0)**: en este modo, el primer elemento del array será la componente 0, es decir tendrá el índice 0.
- ✓ **indexación base-uno(1)**: en este modo, el primer elemento tiene el índice 1.
- ✓ **indexación base-n□**: este modo, es un modo versátil de indexación, en el que el índice del primer elemento puede ser elegido libremente.

En JavaScript cuando trabajamos con índices numéricos utilizaremos la *indexación base-cero(0)*.

Los arrays se introdujeron en JavaScript a partir de la versión 1.1, es decir que en cualquier navegador moderno disponible actualmente no tendrás ningún tipo de problema para poder usar arrays.

http://es.wikipedia.org/wiki/Estructura_de_datos

"Lo que el estilo es a la persona, la estructura es a la obra."

Goytisolo, Luis.

1.1.- Objeto Array.

Un array es una de las mayores estructuras de datos proporcionadas para almacenar y manipular colecciones de datos. A diferencia de otros lenguajes de programación, los arrays en JavaScript son muy versátiles, en el sentido de los diferentes tipos de datos que podemos almacenar en cada posición del array. Ésto nos permite por ejemplo, tener un array de arrays, proporcionando la equivalencia de arrays multidimensionales, pero adaptado a los tipos de datos que necesite nuestra aplicación.

En programación, **un array se define como una colección ordenada de datos**. Lo mejor es que pienses en un array como si fuera una tabla que contiene datos, o también como si fuera una hoja de cálculo.

JavaScript emplea un montón de arrays internamente para gestionar los objetos HTML en el documento, propiedades del navegador, etc. Por ejemplo, si tu documento contiene 10 enlaces, el navegador mantiene una tabla con todos esos enlaces. Tú podrás acceder a esos enlaces por su número de enlace (comenzando en el 0 como primer enlace). Si empleamos la sentencia de array para acceder a esos enlaces, el nombre del array estará seguido del número índice (número del enlace) entre corchetes, como por ejemplo: `document.links[0]` , representará el primer enlace en el documento.

En la unidad anterior si recuerdas, teníamos colecciones dentro del objeto `document`. Pues bien, cada una de esas colecciones (`anchors[]`, `forms[]`, `links[]`, e `images[]`) será un array que contendrá las referencias de todas las anclas, formularios, enlaces e imágenes del documento.

A medida que vayas diseñando tu aplicación, tendrás que identificar las pistas que te permitan utilizar arrays para almacenar datos. Por ejemplo, imagínate que tienes que almacenar un montón de coordenadas geográficas de una ruta a caballo: ésto si que sería un buen candidato a emplear una estructura de datos de tipo array, ya que podríamos asignar nombres a cada posición del array, realizar cálculos, ordenar los puntos, etc. Siempre que veas similitudes con un formato de tabla, será una buena opción para usar un array.

Por ejemplo si tenemos las siguientes variables:

```
var coche1="Seat";
var coche2="BMW";
var coche3="Audi";
var coche4="Toyota";
```

Este ejemplo sería un buen candidato a convertirlo en un array, ya que te permitiría introducir más marcas de coche, sin que tengas que crear nuevas variables para ello. Lo haríamos del siguiente modo:

```
var misCoches=new Array();
misCoches[0]="Seat";
misCoches[1]="BMW";
misCoches[2]="Audi";
misCoches[3]="Toyota";
```

Te explicaremos más en detalle las formas de creación de un array en el siguiente apartado.

A la hora de referenciar las posiciones de un array en JavaScript, que estén definidas de forma numérica, la primera posición de ese array comenzará con el índice 0.

(•) **Verdadero.** ( ) Falso.

1.1.1.- Creación de un array.

Para crear un objeto array, usaremos el constructor **new Array** . Por ejemplo:

```
var miarray= new Array();
```

Un objeto array dispone de una propiedad que nos indica su longitud. Esta propiedad es `length` (longitud, que será 0 para un array vacío). Si queremos precisar el tamaño del array durante la inicialización (por ejemplo para que se cargue con valores `null`), podríamos hacerlo pasándole un parámetro al constructor. Por ejemplo, aquí puedes ver cómo crear un nuevo array que almacene la información de 40 personas:

```
var personas = new Array(40);
```

A diferencia de otros lenguajes de programación, en los que hacer el dimensionamiento previo del array tiene ventajas, en JavaScript no nos dará ningún tipo de ventaja especial, ya que podremos asignar un valor a cualquier posición del array en cualquier momento, esté o no definida previamente. La propiedad `length` se ajustará automáticamente al nuevo tamaño del array. Por ejemplo podríamos hacer una asignación tal como `personas[53]` y eso que nuestro array es de 40 posiciones. En este caso no ocurrirá ningún problema, ya que la propiedad `length` del array se ajustará automáticamente para poder almacenar la posición 53, con lo que la nueva longitud del array será 54, ya que la primera posición del array es la [0] y la última es la [53], tendremos por lo tanto 54 elementos en el array.

```
personas[53] =  "Irene Sáinz Veiga";
longitud = personas.length;      // asignará a la variable longitud el valor 54
```

Introducir datos en un array

Introducir datos en un array, es tan simple como crear una serie de sentencias de asignación, una por cada elemento del array. Ejemplo de un array que contiene el nombre de los planetas del sistema solar:

```
sistemaSolar = new  Array  ;
sistemaSolar[0] = "Mercurio"; sistemaSolar[1] = "Venus"; sistemaSolar[2] = "Tierra";
sistemaSolar[3] = "Marte"; sistemaSolar[4] = "Jupiter"; sistemaSolar[5] = "Saturno";
sistemaSolar[6] = "Urano"; sistemaSolar[7] = "Neptuno";
```

Esta forma es un poco tediosa a la hora de escribir el código, pero una vez que las posiciones del array están cubiertas con los datos, acceder a esa información nos resultará muy fácil:

```
unPlaneta = sistemaSolar[2];     // almacenará en unPlaneta la cadena "Tierra".
```

Otra forma de crear el array puede ser mediante el constructor. En lugar de escribir cada sentencia de asignación para cada elemento, lo podemos hacer creando lo que se denomina un "**array denso**", aportando al **constructor** `array()`, los datos a cubrir separados por comas:

```
sistemaSolar = new array ("Mercurio", "Venus", "Tierra", "Marte", "Jupiter", "Saturno",
"Urano", "Neptuno");
```

El término de "**array denso**" quiere decir que los datos están empaquetados dentro del array, sin espacios y comenzando en la posición 0. Otra forma permitida a partir de la versión de JavaScript 1.2+, sería aquella en la que no se emplea el constructor y se definen los arrays de forma literal:

```
sistemaSolar = ["Mercurio","Venus","Tierra","Marte","Jupiter","Saturno","Urano","Neptuno"];
```

Los corchetes sustituyen a la llamada al constructor `new Array()`. Hay que tener cuidado con esta forma de creación que quizás no funcione en navegadores antiguos.

Y para terminar vamos a ver otra forma de creación de un **array mixto** (o también denominado objeto literal), en el que las posiciones son referenciadas con índices de tipo texto o números, mezclándolos de forma aleatoria. Si las posiciones índice están definidas por un texto, para acceder a ellas lo haremos utilizando su nombre y no su número (en este caso si usamos el número nos daría una posición `undefined`). El formato de creación sería: `nombrearray = { "indice1" : valor , indice2 : "valor" , …}`

(fíjate que en este caso para definir el array de tipo mixto tendremos que hacerlo comenzando y terminando con **llaves** `{ }`). Por ejemplo:

```
var datos = { "numero": 42, "mes"  : "Junio", "hola" : "mundo", 69 : "96" };
document.write("<br/>"+datos["numero"]+ " -- " + datos["mes"]+ " -- " + datos["hola"]+ " -- "
+ datos[69]+ "<br/>");
```

1.1.2.- Recorrido de un array.

Existen múltiples formas de recorrer un array para mostrar sus datos. Veamos algunos ejemplos con el array del sistema Solar:

```
var sistemaSolar = new  Array();
sistemaSolar[0] = "Mercurio";
sistemaSolar[1] = "Venus";
sistemaSolar[2] = "Tierra";
sistemaSolar[3] = "Marte";
sistemaSolar[4] = "Jupiter";
sistemaSolar[5] = "Saturno";
sistemaSolar[6] = "Urano";
sistemaSolar[7] = "Neptuno";
```

Empleando un bucle for, por ejemplo:

```
for (i=0;i<sistemaSolar.length;i++)
     document.write(sistemaSolar[i] + "<br/>");
```

Empleando un bucle while, por ejemplo:

```
var i=0;
while (i < sistemaSolar.length)
{
     document.write(sistemaSolar[i] + "<br/>");
     i++;
}
```

Empleando la sentencia for each in (disponible en versiones de JavaScript 1.6 o superiores):

```
for each (variable en objeto)
     sentencia
```

Esta sentencia repite la variable indicada, sobre todos los valores de las propiedades del objeto. Para cada propiedad distinta, se ejecutará la sentencia especificada.

Ejemplo 1:

```
for each (var planeta in sistemaSolar)
     document.write(planeta+"<br/>");
// Imprimirá todos los nombres de planeta con un salto de línea al final de cada nombre.
```

Ejemplo 2:

```
[document.write(planeta +"<br/>") for each (planeta in sistemaSolar)];
```

"Es curioso, que podamos predecir con siglos de antelación el recorrido de las estrellas más lejanas, y en cambio, no seamos capaces de saber cómo soplará mañana el viento en nuestro pequeño planeta."

SPOERL, Heinrich.

Si accedemos a una posición de un array que no está definida, obtendremos el valor:

- ○ Error en tiempo de ejecución y se detiene la aplicación.
- ◉ **Undefined.**
- ○ No devuelve ningún valor.

1.1.3.- Borrado de elementos en un array.

Para borrar cualquier dato almacenado en un elemento del array, lo podrás hacer ajustando su valor a `null` o a una **cadena vacía** "".

Hasta que apareció el operador **`delete`** en las versiones más modernas de navegadores, no se podía eliminar completamente una posición del array.

Al borrar un elemento del array, se eliminará su índice en la lista de índices del array, pero no se reducirá la longitud del array. Por ejemplo en las siguientes instrucciones:

```
elarray.length;       //     resultado: 8
delete elarray[5];
elarray.length;       //     resultado: 8
elarray[5];           //     resultado: undefined
```

El proceso de borrar una entrada del array no libera la memoria ocupada por esos datos necesariamente. El intérprete de JavaScript, encargado de gestionar la colección de basura en memoria, se encargará de liberar memoria ocupada cuando la necesite.

Si se quiere tener un mayor control sobre la eliminación de elementos de un array, deberías considerar usar el método **`splice(índice, número de elementos a eliminar)`**, que está soportado por la mayoría de los navegadores. Este método se puede usar en cualquier array, y te permite eliminar un elemento o una secuencia de elementos de un array, provocando que la longitud del array se ajuste al nuevo número de elementos.

El operador **`delete`** es compatible a partir de estas versiones : WinIE4 + , MacIE4 + , NN4 + , Moz + , Safari + , Opera + , Chrome +.

Ejemplo de uso del operador `delete`:
Si consideramos el siguiente **array denso**:

```
var  oceanos = new  array("Atlantico","Pacifico","Artico","Indico");
```

Esta clase de array asigna automáticamente índices numéricos a sus entradas, para poder acceder posteriormente a los datos, como por ejemplo con un bucle:

```
for  (var  i=0; i < oceanos.length;  i++)
{
     if  (oceanos[i] == "Atlantico")
     {
          //  instrucciones a realizar..
     }
}
```

Si ejecutamos la instrucción:

```
delete oceanos[2];
```

Se producirán los siguientes cambios: en primer lugar el tercer elemento del array ("Artico"), será eliminado del mismo, pero la longitud del array seguirá siendo la misma, y el array quedará tal y como:

```
oceanos[0] = "Atlantico";
oceanos[1] = "Pacifico";
oceanos[3] = "Indico";
```

Si intentamos referenciar `oceanos[2]` nos devolverá un resultado indefinido (`undefined`).

Si queremos eliminar la tercera posición y que el array reduzca su tamaño podríamos hacerlo con la instrucción:

```
oceanos.splice(2,1); // las posiciones del array resultante serán 0, 1 y 2.
```

El operador `delete`, se recomienda para arrays que usen texto como índices del array, ya que de esta forma se producirán menos confusiones a la hora de borrar los elementos.

1.1.4.- Propiedades y métodos.

Vamos a ver a continuación, las propiedades y métodos que podemos usar con cualquier array que utilicemos en Javascript.

Propiedades del objeto array:

Propiedad	Descripción
constructor	Devuelve la función que creó el prototipo del objeto array.
length	Ajusta o devuelve el número de elementos en un array.
prototype	Te permite añadir propiedades y métodos a un objeto.

Métodos del objeto array:

Métodos	Descripción
concat()	Une dos o más arrays, y devuelve una copia de los arrays unidos.
join()	Une todos los elementos de un array en una cadena de texto.
pop()	Elimina el último elemento de un array y devuelve ese elemento.
push()	Añade nuevos elementos al final de un array, y devuelve la nueva longitud.
reverse()	Invierte el orden de los elementos en un array.
shift()	Elimina el primer elemento de un array, y devuelve ese elemento.
slice()	Selecciona una parte de un array y devuelve el nuevo array.
sort()	Ordena los elementos de un array.
splice()	Añade/elimina elementos a un array.
toString()	Convierte un array a una cadena y devuelve el resultado.
unshift()	Añade nuevos elementos al comienzo de un array, y devuelve la nueva longitud.
valueOf()	Devuelve el valor primitivo de un array.

Ejemplo de algunos métodos del objeto array:

Método reverse():

```
<script type="text/javascript">
     var frutas = ["Plátano", "Naranja", "Manzana", "Melocotón"];
     document.write(frutas.reverse());       // Imprimirá:  Melocotón,Manzana,Naranja,Plátano
</script>
```

Método slice():

```
<script type="text/javascript">
   var frutas = ["Plátano", "Naranja", "Manzana", "Melocotón"];
   document.write(frutas.slice(0,1) + "<br />");    // imprimirá: Plátano
   document.write(frutas.slice(1) + "<br />");      // imprimirá: Naranja,Manzana,Melocotón
   document.write(frutas.slice(-2) + "<br />");     // imprimirá: Manzana, Melocotón
   document.write(frutas + "<br />");       // imprimirá: Plátano,Naranja,Manzana,Melocotón
</script>
```

http://www.w3schools.com/jsref/jsref_obj_array.asp

1.2.- Arrays paralelos.

El usar arrays para almacenar información, facilita una gran versatilidad en las aplicaciones, a la hora de realizar búsquedas de elementos dentro del array. Pero en algunos casos, podría ser muy útil hacer las búsquedas en varios arrays a la vez, de tal forma que podamos almacenar diferentes tipos de información, en arrays que estén sincronizados.

Cuando tenemos dos o más arrays, que utilizan el mismo índice para referirse a términos homólogos, se denominan **arrays paralelos**.

Por ejemplo consideremos los siguientes arrays:

```
var profesores = ["Cristina","Catalina","Vieites","Benjamin"];
var asignaturas=["Seguridad","Bases de Datos","Sistemas Informáticos","Redes"];
var alumnos=[24,17,28,26];
```

Usando estos tres arrays de forma sincronizada, y haciendo referencia a un mismo índice (por ejemplo índice=3), podríamos saber que Benjamín es profesor de Redes y tiene 26 alumnos en clase.

Veamos un ejemplo de código que recorrería todos los profesores que tengamos en el array imprimiendo la información sobre la asignatura que imparten y cuantos alumnos tienen en clase:

```
<script type="text/javascript">
   var profesores = ["Cristina","Catalina","Vieites","Benjamin"];
   var asignaturas=["Seguridad","Bases de Datos","Sistemas Informáticos","Redes"];
   var alumnos=[24,17,28,26];
   function imprimeDatos(indice){
      document.write("<br/>"+profesores[indice]+" del módulo de "+asignaturas[indice]+", tiene
"+alumnos[indice]+" alumnos en clase.");
   }
   for (i=0;i<profesores.length;i++){
      imprimeDatos(i);
   }
</script>
```

Éste será el resultado que obtendremos de la aplicación anterior:

Cristina del módulo de Seguridad, tiene 24 alumnos en clase.
Catalina del módulo de Bases de Datos, tiene 17 alumnos en clase.
Vieites del módulo de Sistemas Informáticos, tiene 28 alumnos en clase.
Benjamin del módulo de Redes, tiene 26 alumnos en clase.

Para que los arrays paralelos sean homogéneos, éstos tendrán que tener la misma longitud, ya que de esta forma se mantendrá la consistencia de la estructura lógica creada.

Entre las ventajas del uso de arrays paralelos tenemos:

- ✓ Se pueden usar en lenguajes que soporten solamente arrays, como tipos primitivos y no registros (como puede ser JavaScript).
- ✓ Son fáciles de entender y utilizar.
- ✓ Pueden ahorrar una gran cantidad de espacio, en algunos casos evitando complicaciones de sincronización.
- ✓ El recorrido secuencial de cada posición del array, es extremadamente rápido en las máquinas actuales.

http://www.webreference.com/programming/javascript/diaries/11/4.html

1.3.- Arrays multidimensionales.

Una alternativa a los arrays paralelos es la simulación de un array multidimensional. Si bien es cierto que en JavaScript los arrays son uni-dimensionales, podemos crear arrays que en sus posiciones contengan otros arrays u otros objetos. Podemos crear de esta forma *arrays bidimensionales, tridimensionales*, etc.

Por ejemplo podemos realizar el ejemplo anterior creando un **array bidimensional** de la siguiente forma:

```
var datos = new Array();
datos[0] = new Array("Cristina","Seguridad",24);
datos[1] = new Array("Catalina","Bases de Datos",17);
datos[2] = new Array("Vieites","Sistemas Informáticos",28);
datos[3] = new Array("Benjamin","Redes",26);
```

ó bien usando una definición más breve y literal del array:

```
var datos = [
          ["Cristina","Seguridad",24],
          ["Catalina","Bases de Datos",17],
          ["Vieites","Sistemas Informáticos",28],
          ["Benjamin","Redes",26]
      ];
```

Para acceder a un dato en particular, de un array de arrays, se requiere que hagamos una doble referencia. La primera referencia, será a una posición del array principal, y la segunda referencia, a una posición del array almacenado dentro de esa casilla del array principal. Ésto se hará escribiendo el nombre del array, y entre corchetes cada una de las referencias: `nombrearray[indice1][indice2][indice3]` (nos permitiría acceder a una posición determinada en un array tridimensional).

Por ejemplo:

```
document.write("<br/>Quien imparte Bases de Datos? "+datos[1][0]); // Catalina
document.write("<br/>Asignatura de Vieites: "+datos[2][1]);        // Sistemas Informaticos
document.write("<br/>Alumnos de Benjamin: "+datos[3][2]);          // 26
```

Si queremos imprimir toda la información del array multidimensional, tal y como hicimos en el apartado anterior podríamos hacerlo con un bucle `for`:

```
for (i=0;i<datos.length;i++){
     document.write("<br/>"+datos[i][0]+" del módulo de "+datos[i][1]+", tiene "+datos[i][2]+"
alumnos en clase.");
}
```

Obtendríamos como resultado:

Cristina del módulo de Seguridad, tiene 24 alumnos en clase.
Catalina del módulo de Bases de Datos, tiene 17 alumnos en clase.
Vieites del módulo de Sistemas Informáticos, tiene 28 alumnos en clase.
Benjamin del módulo de Redes, tiene 26 alumnos en clase.

También podríamos imprimir todos los datos de los arrays dentro de una tabla:

```
document.write("<table border=1>");
for (i=0;i<datos.length;i++){
     document.write("<tr>");
     for (j=0;j<datos[i].length;j++)     {
          document.write("<td>"+datos[i][j]+"</td>");
     }
     document.write("</tr>");
}
document.write("</table>");
```

2.- Creación de funciones.

Caso práctico
__Juan__ está siguiendo los progresos de __Antonio__, y le recomienda empezar a ver funciones. Las funciones son una herramienta muy potente en los lenguajes de programación, ya que le van a permitir realizar tareas de una manera mucho más organizada, y además le permitirán reutilizar un montón de código en sus aplicaciones.
__Antonio__ verá como crear funciones, cómo pasar parámetros, el ámbito de las variables dentro de las funciones, cómo anidar funciones y las funciones predefinidas en JavaScript. __Antonio__ está muy ilusionado con este tema, ya que a lo largo de las unidades anteriores ha estado utilizando funciones y es ahora el momento en el que realmente verá toda la potencia que le pueden aportar a sus aplicaciones.

En unidades anteriores ya has visto y utilizado alguna vez funciones. **Una función es la definición de un conjunto de acciones pre-programadas**. Las funciones se llaman a través de eventos o bien mediante comandos desde nuestro script.

Siempre que sea posible, tienes que diseñar funciones que puedas reutilizar en otras aplicaciones, de esta forma, tus funciones se convertirán en pequeños bloques constructivos que te permitirán ir más rápido en el desarrollo de nuevos programas.

Si conoces otros lenguajes de programación, quizás te suene el término de *subrutina o procedimiento*. En JavaScript no vamos a distinguir entre **procedimientos** (que ejecutan acciones), o **funciones** (que ejecutan acciones y devuelven valores). **En JavaScript siempre se llamarán funciones**.

Una función es capaz de devolver un valor a la instrucción que la invocó, pero ésto no es un requisito obligatorio en JavaScript. Cuando una función devuelve un valor, la instrucción que llamó a esa función, la tratará como si fuera una expresión.

Te mostraré algunos ejemplos en un momento, pero antes de nada, vamos a ver la sintaxis formal de una función:

```
function nombreFunción ( [parámetro1]....[parámetroN] ){
     // instrucciones
}
```

Si nuestra función va a **devolver algún valor** emplearemos la palabra reservada `return`, para hacerlo. Ejemplo:

```
function nombreFunción ( [parámetro1]....[parámetroN] ){
     // instrucciones
     return valor;
}
```

Los nombres que puedes asignar a una función, tendrán las mismas restricciones que tienen los elementos HTML y las variables en JavaScript. Deberías asignarle un nombre que realmente la identifique, o que indique qué tipo de acción realiza. Puedes usar palabras compuestas como `chequearMail` o `calcularFecha`, y fíjate que las funciones suelen llevar un verbo, puesto que las funciones son elementos que realizan acciones.

Una recomendación que te hacemos, es la de que las funciones sean muy específicas, es decir que no realicen tareas adicionales a las inicialmente propuestas en esa función.

Para realizar una llamada a una función lo podemos hacer con:

```
nombreFuncion();// Esta llamada ejecutaría las instrucciones programadas dentro de la función.
```

Otro ejemplo de uso de una función en una asignación:

```
variable=nombreFuncion();     // En este caso la función devolvería un valor que se asigna a la variable.
```

Las funciones en JavaScript también son objetos, y como tal tienen métodos y propiedades. Un método, aplicable a cualquier función puede ser `toString()`, el cual nos devolverá el código fuente de esa función.

2.1.- Parámetros.

Cuando se realiza una llamada a una función, muchas veces es necesario pasar parámetros (también conocidos como argumentos). Este mecanismo nos va a permitir enviar datos entre instrucciones.

Para pasar parámetros a una función, tendremos que escribir dichos parámetros entre paréntesis y separados por comas.

A la hora de definir una función que recibe parámetros, lo que haremos es, escribir los nombres de las variables que recibirán esos parámetros entre los paréntesis de la función.

Veamos el siguiente ejemplo:

```
function saludar(a,b){
     alert("Hola " + a + " y "+ b +".");
}
```

Si llamamos a esa función desde el código:

```
saludar("Martin","Silvia");      //Mostraría una alerta con el texto: Hola Martin y Silvia.
```

Los parámetros que usamos en la definición de la función a y b, no usan la palabra reservada `var` para inicializar dichas variables. Esos *parámetros a y b serán variables locales a la función*, y se inicializarán automáticamente en el momento de llamar a la función, con los valores que le pasemos en la llamada. En el siguiente apartado entraremos más en profundidad en lo que son las variables locales y globales.

Otro ejemplo de función que devuelve un valor:

```
function devolverMayor(a,b){
     if (a > b) then
          return a;
     else
          return b;
}
```

Ejemplo de utilización de la función anterior:

```
document.write ("El número mayor entre 35 y 21 es el: " + devolverMayor(35,21) + ".");
```

"La inteligencia es la función que adapta los medios a los fines."

HARTMANN, N.

http://www.ulpgc.es/otros/tutoriales/JavaScript/cap5.htm
http://www.desarrolloweb.com/articulos/585.php

2.2.- Ámbito de las variables.

Ha llegado la hora de distinguir entre las variables que se definen fuera de una función, y las que se definen dentro de las funciones.

Las variables que se definen fuera de las funciones se llaman **variables globales.** Las **variables** que se definen dentro de las funciones, con la palabra reservada `var`, se llaman variables locales.

Una **variable global** en JavaScript tiene una connotación un poco diferente, comparando con otros lenguajes de programación. Para un script de JavaScript, el alcance de una variable global, se limita al documento actual que está cargado en la ventana del navegador o en un frame. Sin embargo cuando inicializas una variable como variable global, quiere decir que todas las instrucciones de tu script (incluidas las instrucciones que están dentro de las funciones), tendrán acceso directo al valor de esa variable. Todas las instrucciones podrán leer y modificar el valor de esa variable global.

En el momento que una página se cierra, todas las variables definidas en esa página se eliminarán de la memoria para siempre. Si necesitas que el valor de una variable persista de una página a otra, tendrás que utilizar técnicas que te permitan almacenar esa variable (como las cookies, o bien poner esa variable en el documento frameset, etc.).

Aunque el uso de la palabra reservada `var`, para inicializar variables es opcional, te recomiendo que la uses ya que así te protegerás de futuros cambios en las próximas versiones de JavaScript.

En contraste a las variables globales, una **variable local será definida dentro de una función**. Antes viste que podemos definir variables en los parámetros de una función (sin usar `var`), pero también podrás definir nuevas variables dentro del código de la función. En este caso, **si que se requiere el uso de la palabra reservada `var` cuando definimos una variable local**, ya que de otro modo, esta variable será reconocida como una variable global.

El alcance de una variable local está solamente dentro del ámbito de la función. Ninguna otra función o instrucciones fuera de la función podrán acceder al valor de esa variable.

Reutilizar el nombre de una variable global como local es uno de los bugs (*fallo en programación. Generalmente se refiere a fallos o errores de tipo lógico, que suelen ser más difíciles de detectar que los errores sintácticos, los cuáles si que son mostrados por el analizador sintáctico del lenguaje*) más sutiles y por consiguiente más difíciles de encontrar en el código de JavaScript. La variable local en momentos puntuales ocultará el valor de la variable global, sin avisarnos de ello. Como recomendación, no reutilices un nombre de variable global como local en una función, y tampoco declares una variable global dentro de una función, ya que podrás crear fallos que te resultarán difíciles de solucionar.

Ejemplo de variables locales y globales:

```
/// Uso de variables locales y globales no muy recomendable, ya que estamos empleando el mismo
nombre de variable en global y en local.
var  chica  =  "Aurora";                    // variable global
var  perros  =  "Lacky, Samba y Ronda";     // variable global
function  demo(){
   //  Definimos una variable local (fíjate que es obligatorio para las variables locales usar
var)
   //  Esta variable local tendrá el mismo nombre que otra variable global pero con distinto
contenido.
   // Si no usáramos var estaríamos modificando la variable global chica.
   var  chica  =  "Raquel";                 // variable local
   document.write( "<br/>"+ perros +"  no pertenecen a "+ chica +".");
}
// Llamamos a la función para que use las variables locales.
demo();
//  Utilizamos las variables globales definidas al comienzo.
document.write( "<br/>"+ perros +"  pertenecen a "+ chica +".");
```

Como resultado obtenemos:

Lacky, Samba y Ronda no pertenecen a Raquel.
Lacky, Samba y Ronda pertenecen a Aurora.

2.3.- Funciones anidadas.

Los navegadores más modernos nos proporcionan la opción de anidar unas funciones dentro de otras. Es decir podemos programar una función dentro de otra función.

Cuando no tenemos funciones anidadas, cada función que definamos será accesible por todo el código, es decir serán funciones globales. Con las funciones anidadas, podemos encapsular la accesibilidad de una función dentro de otra y hacer que esa función sea privada o local a la función principal. Tampoco te recomiendo el reutilizar nombres de funciones con esta técnica, para evitar problemas o confusiones posteriores.

La estructura de las funciones anidadas será algo así:

```
function  principalA(){
   // instrucciones
   function  internaA1(){
      // instrucciones
   }
   // instrucciones
}
function  principalB(){
   // instrucciones
   function  internaB1(){
      // instrucciones
   }
   function  internaB2(){
      // instrucciones
   }
   // instrucciones
}
```

Una buena opción para aplicar las funciones anidadas, es cuando tenemos una secuencia de instrucciones que necesitan ser llamadas desde múltiples sitios dentro de una función, y esas instrucciones sólo tienen significado dentro del contexto de esa función principal. En otras palabras, en lugar de romper la secuencia de una función muy larga en varias funciones globales, haremos lo mismo pero utilizando funciones locales.

Ejemplo de una función anidada:

```
function hipotenusa(a, b){
   function cuadrado(x){
          return x*x;
     }
   return Math.sqrt(cuadrado(a) + cuadrado(b));
}
document.write("<br/>La hipotenusa de 1 y 2 es: "+hipotenusa(1,2));
// Imprimirá: La hipotenusa de 1 y 2 es: 2.23606797749979
```

"La función última de la crítica es que satisfaga la función natural de desdeñar, lo que conviene a la buena higiene del espíritu."

PESSOA, Fernando

2.4.- Funciones predefinidas del lenguaje.

Has visto en unidades anteriores un montón de objetos con sus propiedades y métodos. Si te das cuenta todos los métodos en realidad son funciones (llevan siempre paréntesis con o sin parámetros). Pues bien en JavaScript, disponemos de algunos elementos que necesitan ser tratados a escala global y que no pertenecen a ningún objeto en particular (o que se pueden aplicar a cualquier objeto).

Propiedades globales en JavaScript:

Propiedad	Descripción
Infinity	Un valor numérico que representa el infinito positivo/negativo.
NaN	Valor que no es numérico "Not a Number".
undefined()	Indica que a esa variable no le ha sido asignado un valor.

Te mostraremos aquí una lista de funciones predefinidas, que se pueden utilizar a nivel global en cualquier parte de tu código de JavaScript. Estas funciones no están asociadas a ningún objeto en particular. Típicamente, estas funciones te permiten convertir datos de un tipo a otro tipo.

Funciones globales o predefinidas en JavaScript:

Función	Descripción
decodeURI()	Decodifica los caracteres especiales de una URL excepto: , / ? : @ & = + $ #
decodeURIComponent()	Decodifica todos los caracteres especiales de una URL.
encodeURI()	Codifica los caracteres especiales de una URL excepto: , / ? : @ & = + $ #
encodeURIComponent()	Codifica todos los caracteres especiales de una URL.
escape()	Codifica caracteres especiales en una cadena, excepto: * @ - _ + . /
eval()	Evalúa una cadena y la ejecuta si contiene código u operaciones.
isFinite()	Determina si un valor es un número finito válido.
isNaN()	Determina cuando un valor no es un número.
Number()	Convierte el valor de un objeto a un número.
parseFloat()	Convierte una cadena a un número real.
parseInt()	Convierte una cadena a un entero.
unescape()	Decodifica caracteres especiales en una cadena, excepto: * @ - _ + . /

Ejemplo de la **función** `eval()`:

```
<script type="text/javascript">
     eval("x=50;y=30;document.write(x*y)");          // Imprime 1500
     document.write("<br />" + eval("8+6"));         // Imprime 14
     document.write("<br />" + eval(x+30));          // Imprime 80
</script>
```

El siguiente enlace amplía información sobre las funciones predefinidas en JavaScript.
http://www.w3schools.com/jsref/jsref_obj_global.asp

3.- Creación de objetos a medida.

Caso práctico
Antonio *ha hecho una pausa para tomar café, y charla con* ***Juan*** *de todo lo que ha aprendido sobre las funciones. Comenta lo interesante que ha sido, pero le surge una duda, ya que cuando estudió los objetos en JavaScript, vio que tenían propiedades y métodos, y los métodos tienen el mismo formato que las funciones que acaba de estudiar.* ***Juan*** *le dice que efectivamente eso es así, porque en realidad los métodos son funciones que se asignan a un objeto determinado, y que a parte de los objetos que ya ha estudiado en unidades anteriores, en JavaScript podrá crear sus propios objetos, con los métodos (funciones) y propiedades que él quiera.*
Terminan su café y los dos vuelven al despacho para buscar documentación sobre cómo crear nuevos objetos en JavaScript.

En unidades anteriores has visto, como toda la información que proviene del navegador o del documento está organizada en un modelo de objetos, con propiedades y métodos. Pues bien, JavaScript también te da la oportunidad de crear tus propios objetos en memoria, objetos con propiedades y métodos que tú puedes definir a tu antojo. Estos objetos no serán elementos de la página de interfaz de usuario, pero sí que serán objetos que podrán contener **datos (propiedades) y funciones (métodos)**, cuyos resultados si que se podrán mostrar en el navegador. El definir tus propios objetos, te permitirá enlazar a cualquier número de propiedades o métodos que tú hayas creado para ese objeto. Es decir, tú controlarás la estructura del objeto, sus datos y su comportamiento.

También hay que dejar claro que, **JavaScript no es un lenguaje orientado a objetos de verdad en sentido estricto**. Se considera que, JavaScript **es un lenguaje basado en objetos**. La diferencia entre orientado a objetos y basado en objetos es significante, y tiene que ver sobre todo en cómo los objetos se pueden extender. Quizás en un futuro no muy lejano podamos ver que JavaScript soporte clases, interfaces, herencia, etc.

Un objeto en JavaScript es realmente una colección de propiedades. Las propiedades pueden tener forma de datos, tipos, funciones (métodos) o incluso otros objetos. De hecho sería más fácil de entender un objeto como un array de valores, cada uno de los cuales está asociado a una propiedad (un tipo de datos, método u objeto). Un momento: ¿un método puede ser una propiedad de un objeto? Pues en JavaScript parece que sí.
Una función contenida en un objeto se conoce como un método. Los métodos no son diferentes de las funciones que has visto anteriormente, excepto que han sido diseñados para ser utilizados en el contexto de un objeto, y por lo tanto, tendrán acceso a las propiedades de ese objeto. Esta conexión entre propiedades y métodos es uno de los ejes centrales de la orientación a objetos.

Los objetos se crean empleando una función especial denominada **constructor**, determinada por el nombre del objeto. Ejemplo de una función constructor:

```
function Coche(){
     // propiedades y métodos
}
```

Aunque esta función no contiene código, es sin embargo la base para crear objetos de tipo `Coche`. Puedes pensar en un constructor como un anteproyecto o plantilla, que será utilizada para crear objetos. Por convención, los nombres de los constructores se ponen generalmente con las iniciales de cada palabra en mayúscula, y cuando creamos un objeto con ese constructor (**instancia de ese objeto**), lo haremos empleando minúsculas al principio. Por ejemplo: `var unCoche = new Coche();`
La palabra reservada **new se emplea para crear objetos en JavaScript**. Al crear la variable `unCoche`, técnicamente podríamos decir que hemos creado una instancia de la clase Coche, o que hemos instanciado el objeto `Coche`, o que hemos creado un objeto `Coche`, etc. Es decir hay varias formas de expresarlo, pero todas quieren decir lo mismo.

http://www.desarrolloweb.com/articulos/787.php
http://www.desarrolloweb.com/articulos/788.php

3.1.- Definición de propiedades.

Una vez que ya sabemos como crear un constructor para un objeto, vamos a ver cómo podemos crear una propiedad específica para ese objeto. **Las propiedades para nuestro objeto se crearán dentro del constructor** empleando para ello la palabra reservada `this`. Véase el siguiente ejemplo:

```
function Coche(){
     // Propiedades
     this.marca =  "Audi A6";
     this.combustible =  "diesel";
     this.cantidad = 0;          // Cantidad de combustible en el depósito.
}
```

La palabra reservada `this`, se utiliza **para hacer referencia al objeto actual**, que en este caso será el objeto que está siendo creado por el constructor. Por lo tanto, usarás `this`, para crear nuevas propiedades para el objeto. El único problema con el ejemplo anterior es, que todos los coches que hagamos del tipo `Coche` será siempre Audi A6, diésel, y sin litros de combustible en el depósito. Por ejemplo;

```
var cocheDeMartin = new Coche();
var cocheDeSilvia = new Coche();
```

A partir de ahora, si no modificamos las propiedades del coche de Martin y de Silvia, en el momento de instanciarlos tendrán ambos un Audi A6 a diesel y sin combustible en el depósito.

Lo ideal sería por lo tanto que en el momento de instanciar un objeto de tipo Coche, que le digamos al menos la marca de coche y el tipo de combustible que utiliza. Para ello tenemos que modificar el constructor, que quedará de la siguiente forma:

```
function Coche(marca, combustible){
     // Propiedades
     this.marca =  marca;
     this.combustible =  combustible;
     this.cantidad = 0;       // Cantidad de combustible inicial por defecto en el depósito.
}
```

Ahora sí que podríamos crear dos tipos diferentes de coche:

```
var cocheDeMartin = new Coche("Volkswagen Golf","gasolina");
var cocheDeSilvia = new Coche("Mercedes SLK","diesel");
```

Y también podemos acceder a las propiedades de esos objetos, consultar sus valores o modificarlos. Por ejemplo:

```
document.write("<br/>El coche de Martin es un: "+cocheDeMartin.marca+" a
"+cocheDeMartin.combustible);
document.write("<br/>El coche de Silvia es un: "+cocheDeSilvia.marca+" a "+
cocheDeSilvia.combustible);
// Imprimirá:
// El coche de Martin es un: Volkswagen Golf a gasolina
// El coche de Silvia es un: Mercedes SLK a diesel
// Ahora modificamos la marca y el combustible del coche de Martin:
cocheDeMartin.marca = "BMW X5";
cocheDeMartin.combustible = "diesel";
document.write("<br/>El coche de Martin es un: " + cocheDeMartin.marca + " a " +
cocheDeMartin.combustible);
// Imprimirá: El coche de Martin es un: BMW X5 a diesel
```

3.2.- Definición de métodos.

Las propiedades son solamente la mitad de la ecuación de la Orientación a Objetos en JavaScript. La otra mitad son **los métodos, que serán funciones que se enlazarán a los objetos**, para que dichas funciones puedan acceder a las propiedades de los mismos.

Nos definimos un ejemplo de método, que se podría utilizar en la clase Coche:

```
function rellenarDeposito (litros){
     // Modificamos el valor de la propiedad cantidad de combustible
     this.cantidad = litros;
}
```

Fíjate que el método **rellenarDeposito**, que estamos programando a nivel global, hace referencia a la propiedad `this.cantidad` para indicar cuantos litros de combustible le vamos a echar al coche. Lo único que faltaría aquí es realizar la conexión entre el método **rellenarDeposito** y el objeto de tipo Coche (recuerda que los objetos podrán tener propiedades y métodos y hasta este momento sólo hemos definido propiedades dentro del constructor). Sin esta conexión la palabra reservada **this** no tiene sentido en esa función, ya que no sabría cuál es el objeto actual. Veamos cómo realizar la conexión de ese método, con el objeto dentro del constructor:

```
function Coche(marca, combustible){
     // Propiedades
     this.marca =  marca;
     this.combustible =  combustible;
     this.cantidad = 0;       // Cantidad de combustible inicial por defecto en el depósito.
     // Métodos
     this.rellenarDeposito = rellenarDeposito;
}
```

Aquí se ve de forma ilustrada, que los métodos son en realidad propiedades: se declaran igual que las propiedades, por lo que son enmascarados como propiedades en JavaScript. Hemos creado una nueva propiedad llamada **rellenarDeposito** y se le ha asociado el método **rellenarDeposito**. Es muy importante destacar que el método **rellenarDeposito()** se referencia sin paréntesis dentro del constructor, **this.rellenarDeposito = rellenarDeposito**.

Ejemplo de uso del método anterior:

```
cocheDeMartin.rellenarDeposito(35);
document.write("<br/>El coche de Martin tiene "+cocheDeMartin.cantidad+ " litros de " +
cocheDeMartin.combustible+ " en el depósito.");
// Imprimirá
// El coche de Martin tiene 35 litros de diesel en el depósito.
```

La forma en la que hemos definido el método **rellenarDeposito** a nivel global, no es la mejor práctica en la programación orientada a objetos. **Una mejor aproximación sería definir el contenido de la función rellenarDeposito dentro del constructor, ya que de esta forma los métodos al estar programados a nivel local aportan mayor privacidad y seguridad al objeto en general**, por ejemplo:

```
function Coche(marca, combustible){
     // Propiedades
     this.marca =  marca;
     this.combustible =  combustible;
     this.cantidad = 0;
     // Métodos
     this.rellenarDeposito = function (litros){
          this.cantidad=litros;
     };
}
```

3.3.- Definición de objetos literales.

Otra forma de definir objetos es hacerlo de forma literal.

Un literal es un valor fijo que se especifica en JavaScript. Un objeto literal será un conjunto, de cero o más parejas del tipo **nombre:valor**.

Por ejemplo:

```
avion = { marca:"Boeing" , modelo:"747" , pasajeros:"450" };
```

Es equivalente a:

```
var avion = new Object();
avion.marca = "Boeing";
avion.modelo = "747";
```

```
avion.pasajeros = "450";
```

Para referirnos desde JavaScript a una propiedad del objeto avión podríamos hacerlo con:

```
document.write(avion.marca);  // o trambién se podría hacer con:
document.write(avion["modelo"]);
```

Podríamos tener un conjunto de objetos literales simplemente creando un array que contenga en cada posición una definición de objeto literal:

```
var datos=[
{"id":"2","nombrecentro":"IES A Piringalla" ,"localidad":"Lugo","provincia":"Lugo"},
{"id":"10","nombrecentro":"IES As Fontiñas","localidad":"Santiago","provincia":"A Coruña"},
{"id":"9","nombrecentro":"IES As Lagoas","localidad":"Ourense","provincia":"Ourense"},
{"id":"8","nombrecentro":"IES Cruceiro Baleares","localidad":"Culleredo","provincia":"A
Coruña"},
{"id":"6","nombrecentro":"IES Cruceiro Baleares","localidad":"Culleredo","provincia":"A
Coruña"},{"id":"4","nombrecentro":"IES de Teis","localidad":"Vigo","provincia":"Pontevedra"},
{"id":"5","nombrecentro":"IES Leliadoura","localidad":"Ribeira","provincia":"A Coruña"},
{"id":"7","nombrecentro":"IES Leliadoura","localidad":"Ribeira","provincia":"A Coruña"},
{"id":"1","nombrecentro":"IES Ramon Aller
Ulloa","localidad":"Lalin","provincia":"Pontevedra"},
{"id":"3","nombrecentro":"IES San Clemente","localidad":"Santiago de
Compostela","provincia":"A Coruña"}
];
```

De la siguiente forma se podría recorrer el array de datos para mostrar su contenido:

```
for (var i=0; i< datos.length; i++){
     document.write("Centro ID: "+datos[i].id+" - ");
     document.write("Nombre: "+datos[i].nombrecentro+" - ");
     document.write("Localidad: "+datos[i].localidad+" - ");
     document.write("Provincia: "+datos[i].provincia+"<br/>");
}
```

Y obtendríamos como resultado:

```
Centro ID: 2 - Nombre: IES A Piringalla - Localidad: Lugo - Provincia: Lugo
Centro ID: 10 - Nombre: IES As Fontiñas - Localidad: Santiago - Provincia: A Coruña
Centro ID: 9 - Nombre: IES As Lagoas - Localidad: Ourense - Provincia: Ourense
Centro ID: 8 - Nombre: IES Cruceiro Baleares - Localidad: Culleredo - Provincia: A Coruña
Centro ID: 6 - Nombre: IES Cruceiro Baleares - Localidad: Culleredo - Provincia: A Coruña
Centro ID: 4 - Nombre: IES de Teis - Localidad: Vigo - Provincia: Pontevedra
Centro ID: 5 - Nombre: IES Leliadoura - Localidad: Ribeira - Provincia: A Coruña
Centro ID: 7 - Nombre: IES Leliadoura - Localidad: Ribeira - Provincia: A Coruña
Centro ID: 1 - Nombre: IES Ramon Aller Ulloa - Localidad: Lalin - Provincia: Pontevedra
Centro ID: 3 - Nombre: IES San Clemente - Localidad: Santiago de Compostela - Provincia: A
Coruña
```

TEMA 5

Contenido

Gestión de eventos y formularios en JavaScript.

Caso práctico
***Antonio**, afronta una de las partes más interesantes en su estudio de JavaScript. Se trata de la gestión de los eventos en JavaScript y cómo validar formularios.*
***Antonio** estaba deseando llegar a esta sección, ya que su trabajo de desarrollo en el proyecto se va a centrar en muchas de las cosas que va a ver ahora. Va a tener que validar todos los formularios de la web antigua y tendrá que hacerlo con JavaScript, dando mensajes al usuario, sobre los posibles errores que se vaya encontrando durante la validación. La validación de un formulario es primordial, ya que lo que se busca es que cuando los datos sean enviados al servidor, vayan lo más coherentes posible.*
*Gracias a la gestión de eventos, **Antonio** podrá por ejemplo, capturar acciones del usuario cuando pulse un botón o cuando pase el ratón por zonas del documento, al introducir datos, etc.*
***Juan** está también muy ilusionado con los progresos de **Antonio** durante todo este tiempo, y le facilita toda la documentación y algunos ejemplos interesantes de validaciones de datos con JavaScript.*

1.- El objeto Form.

Caso práctico
Los formularios, son el principal medio de introducción de datos en una aplicación web, y el principal punto de interactividad con el usuario.
Es aquí donde JavaScript, nos va a permitir aportar toda esa interactividad a los elementos estáticos de un formulario HTML.
Es muy importante conocer al detalle todos los objetos y métodos de los formularios, ya que un porcentaje muy alto de la interactividad que se produce en una página web, proviene del uso de formularios.
*La forma de acceso a un formulario, cómo referenciarlo, su estructura desde el punto de vista de JavaScript y su contenido, será lo que **Antonio** estudiará en este momento.*

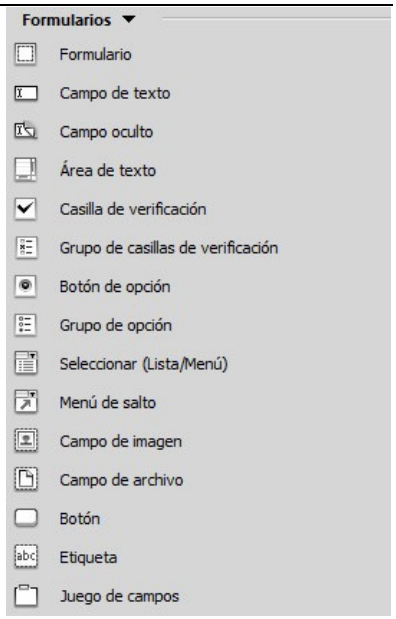
La mayor parte de interactividad entre una página web y el usuario tiene lugar a través de un formulario. Es ahí donde nos vamos a encontrar con los campos de texto, botones, checkboxes, listas, etc. en los que el usuario introducirá los datos, que luego se enviarán al servidor.

En este apartado verás cómo identificar un formulario y sus objetos, cómo modificarlos, cómo examinar las entradas de usuario, enviar un formulario, validar datos, etc.

Los formularios y sus controles, son objetos del DOM que tienen propiedades únicas, que otros objetos no poseen. Por ejemplo, un formulario tiene una propiedad `action`, que le indica al navegador donde tiene que enviar las entradas del usuario, cuando se envía el formulario. Un control `select` posee una propiedad llamada `selectedIndex`, que nos indica la opción de ese campo que ha sido seleccionada por el usuario.

JavaScript añade a los formularios dos características muy interesantes:
- ✓ JavaScript permite examinar y validar las entradas de usuario directamente, en el lado del cliente.
- ✓ JavaScript permite dar mensajes instantáneos, con información de la entrada del usuario.

El objeto de JavaScript `Form`, es una propiedad del objeto `document`. Se corresponderá con la etiqueta `<FORM>` del HTML. Un formulario podrá ser enviado llamando al método `submit` de JavaScript, o bien haciendo click en el botón `submit` del formulario.



En la siguiente imagen, se puede ver un ejemplo de una lista de los objetos utilizados al trabajar con formularios. En este caso, se muestra una captura de la aplicación Dreamweaver de Adobe Systems Incorporated:

"Los errores causados por los datos inadecuados son mucho menores que los que se deben a la total ausencia de datos."
BABBAGE, Charles

¿Si, por ejemplo, usamos JavaScript para validar un formulario, será necesario también validar esos datos en el lado del servidor?
¿Qué pasaría con nuestras validaciones si por ejemplo desactivamos JavaScript en el navegador?
¿Y qué pasaría si alguien programa una copia de nuestro formulario en otro servidor web para enviar datos a nuestro servidor sin validarlos previamente?

1.1.- Formas de selección del objeto Form.

Dentro de un documento tendremos varias formas de selección de un formulario.

Si partimos del siguiente ejemplo:

```
<div  id="menulateral">
     <form  id="contactar" name="contactar" action="...">...</form>
</div>
...
```

Tendremos los siguientes métodos de selección del objeto `Form` en el documento:

Método 1

A través del método `getElementById()` del DOM, nos permite acceder a un objeto a través de su atributo ID. Tendremos que tener la precaución de asignar id únicos a nuestros objetos, para evitar que tengamos objetos con id repetidos.

Ejemplo:

```
var formulario=document.getElementById("contactar");
```

Método 2

A través del método `getElementsByTagName()` del DOM, el cual nos permite acceder a un objeto a través de la etiqueta HTML que queramos. Por ejemplo para acceder a los objetos con etiqueta `form` haremos:

```
var formularios = document.getElementsByTagName("form");
var primerFormulario = formularios[0];     // primer formulario del documento
```

o también todo en una única línea:

```
var primerFormulario = document.getElementsByTagName("form")[0] ;
```

Otra posibilidad interesante que te permite el método anterior, es la de buscar objetos con un padre determinado, por ejemplo:

```
var menu=document.getElementById("menulateral");
var  formularios=menu.getElementsByTagName("form");  //  formularios  contenidos  en  el  menu
lateral
var primerFormulario= formularios[0];     // primer formulario en el menú lateral
```

Método 3

Otro método puede ser a través de la colección `forms[]` del **objeto** `document`. Esta colección es un array, que contiene la referencia a todos los formularios que tenemos en nuestro documento.

Por ejemplo:

```
var formularios = document.forms;       // la referencia a todos los formularios del documento
var miformulario = formularios[0];  // primer formulario del documento
```

o bien:

```
var miformulario = document.forms[0];     // primer formulario del documento
```

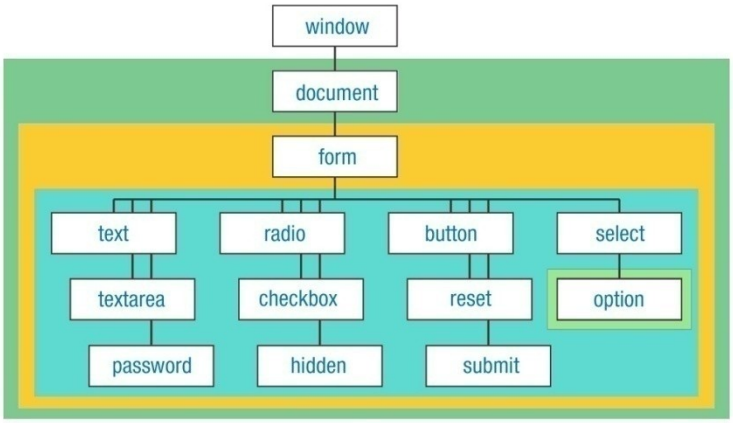
o bien:

```
var miformulario = formularios["contactar"]; // referenciamos al formulario con name "contactar"
```

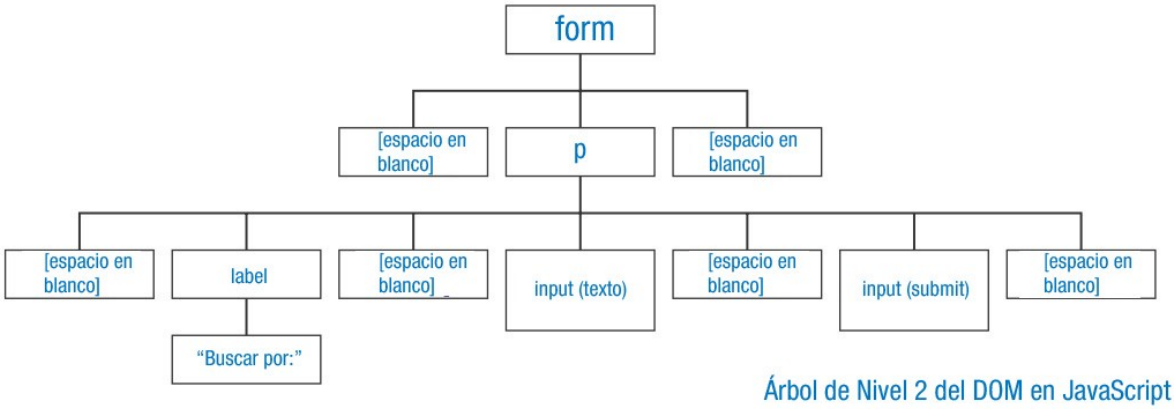
1.2.- El formulario como objeto y contenedor.

Debido a que el DOM ha ido evolucionando con las nuevas versiones de JavaScript, nos encontramos con que el objeto `Form` está dentro de dos árboles al mismo tiempo. En las nuevas definiciones del DOM, se especifica que `Form` es el padre de todos sus nodos hijos, incluidos objetos y textos, mientras que en las versiones antiguas `Form` sólo era padre de sus objetos (`input`, `select`, `button` y elementos `textarea`).

Jerarquía de nivel 0 del DOM para formularios y controles:



Te mostraré con un ejemplo cómo sería el árbol de nivel 2 del DOM para un formulario típico:

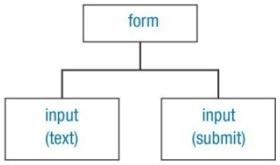


Árbol de Nivel 2 del DOM en JavaScript

Partimos del siguiente código de ejemplo:

```
<form  action="buscar.php" name="elFormulario" id="miFormulario" method="post"> por
<p>
<label  for="busqueda">Buscar por:</label>
<input  id="busqueda"  name="busqueda"  type="text"  value="">
<input  id="submit"  type="submit"  value="Buscar">
</p>
</form>
```

Árbol de nivel 0 para el mismo formulario:



Los dos árboles que te mostré anteriormente pueden ser útiles para diferentes propósitos. El árbol del DOM de nivel 2, se puede utilizar para leer y escribir en todo el documento con un nivel muy fino de granuralidad (*especificidad con la que se define un nivel de detalle*). El árbol del DOM de nivel 0, hace muchísimo más fácil leer y escribir los controles del formulario.

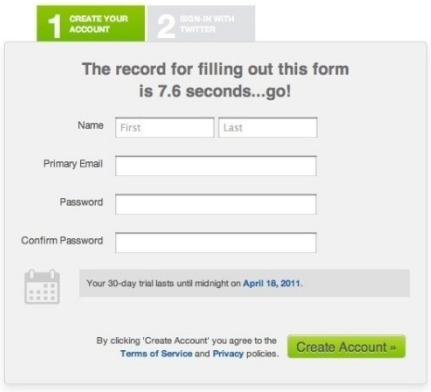
Tienes que darte cuenta que, aunque utilices las técnicas del DOM 0 o DOM 2, los objetos siguen siendo los mismos. Por ejemplo:

```
var elFormulario = document.getElementById("miFormulario");
var control = elFormulario.busqueda;
```

"Son las palabras y las fórmulas, más que la razón, las que crean la mayoría de nuestras opiniones."
LE BON, Gustave

1.3.- Acceso a propiedades y métodos del formulario.

Los formularios pueden ser creados a través de las etiquetas HTML, o utilizando JavaScript y métodos del DOM. En cualquier caso se pueden asignar atributos como `name`, `action`, `target` y `enctype`. Cada uno de estos atributos es una propiedad del objeto `Form`, a las que podemos acceder utilizando su nombre en minúsculas, por ejemplo:




```
var paginaDestino = objetoFormulario.action;
```

Para modificar una de estas propiedades lo haremos mediante asignaciones, por ejemplo:

```
objetoFormulario.action = "http://www.educacion.gob.es/recepcion.php";
```

Estas dos instrucciones las podemos recomponer usando referencias a objetos:

```
var paginaDestino = document.getElementByid("id").action;
document.forms[0].action = "http://www.educacion.gob.es/recepcion.php";
```

Propiedades del objeto Form		
Propiedad	**Descripción**	**W3C**
acceptCharset	Ajusta o devuelve el valor del atributo `accept-charset` en un formulario.	**Sí**
action	Ajusta o devuelve el valor del atributo `action` en un formulario.	**Sí**
enctype	Ajusta o devuelve el valor del atributo `enctype` en un formulario.	**Sí**
length	Devuelve el número de elementos en un formulario.	**Sí**
method	Ajusta o devuelve el valor del atributo `method` en un formulario.	**Sí**
name	Ajusta o devuelve el valor del atributo `name` en un formulario.	**Sí**
target	Ajusta o devuelve el valor del atributo `target` en un formulario.	**Sí**

Métodos del objeto Form		
Método	**Descripción**	**W3C**
reset()	Resetea un formulario.	**Sí**
submit()	Envía un formulario.	**Sí**

Propiedad form.elements[]

La propiedad `elements[]` de un formulario es una colección, que contiene todos los objetos `input` dentro de un formulario. Esta propiedad es otro array, con todos los campos `input` en el orden en el cual aparecen en el código fuente del documento.

Generalmente, es mucho más eficaz y rápido referenciar a un elemento individual usando su ID, pero a veces, los scripts necesitan recorrer cada elemento del formulario, para comprobar que se han introducido sus valores correctamente.
Por ejemplo, empleando la propiedad **`elements[]`**, podemos hacer un bucle que recorra un formulario y si los campos son de tipo texto, pues que los ponga en blanco:

```
var miFormulario = document.getElementById("contactar");
// guardamos la referencia del formulario en una variable.

if (! miFormulario) return false; // Si no existe ese formulario devuelve false.

for (var i=0; i< miFormulario.elements.length; i++){
     if (miFormulario.elements[i].type == "text"){
          miFormulario.elements[i].value = "";
     }
}
```

2.- Objetos relacionados con formularios.

Caso práctico
Una vez visto cómo referenciar a un formulario en JavaScript, tenemos que saber cómo acceder a cada uno de los elementos u objetos, que contiene ese formulario.
Cada uno de los elementos de un formulario, son objetos en JavaScript que tendrán propiedades y métodos, que nos permitirán realizar acciones sobre ellos. Gracias a esos métodos y propiedades, podremos realizar acciones como validar el contenido de un formulario, marcar o desmarcar una determinada opción, mostrar contenido de un campo u ocultarlo, etc.
***Juan** le indica a **Antonio**, que es muy importante que comprenda esta parte, ya que es la base de los procesos de validación que estudiará más adelante, y es una de las tareas que más veces va a tener que realizar en su proyecto de actualización de la web.*

Para poder trabajar con los objetos de un formulario, lo primero que necesitas saber es, cómo referenciar a ese objeto. Eso puedes hacerlo directamente a través de su ID, o bien con su nombre de etiqueta, empleando para ello los métodos del DOM nivel 2. O también se puede hacer usando la sintaxis del DOM nivel 0, y construyendo la jerarquía que comienza por `document`, luego el formulario y finalmente el control.

Lo mejor es identificar cada uno de los objetos con un atributo `id` que sea único, y que no se repita en el documento, así para acceder a cualquier objeto dentro de nuestro documento o formulario lo haremos con:

```
document.getElementById("id-del-control")
```

o

```
document.nombreFormulario.name-del-control
```

Por ejemplo, si consideramos un ejemplo sencillo de formulario:

```
<form  id="formularioBusqueda"  action="cgi-bin/buscar.pl">
   <p>
      <input  type="text"  id="entrada"  name="cEntrada">
      <input  type="submit"  id="enviar"  name="enviar"  value="Buscar...">
   </p>
</form>
```

Las siguientes referencias al campo de texto entrada, serán todas válidas:

```
document.getElementById("entrada")
document.formularioBusqueda.cEntrada
document.formularioBusqueda.elements[0]
document.forms["formularioBusqueda"].elements["cEntrada"]
document.forms["formularioBusqueda"].cEntrada
```

Aunque muchos de los controles de un formulario tienen propiedades en común, algunas propiedades son únicas a un control en particular. Por ejemplo, en un objeto `select` tienes propiedades que te permiten conocer la opción que está actualmente seleccionada. Al igual que los `checkboxes` o los botones de tipo `radio`, que también disponen de propiedades para saber cuál es la opción que está actualmente seleccionada.

A la hora de identificar los objetos en un formulario lo más recomendable es que el atributo id y el atributo name sean iguales y que no se repitan los id en todo el documento:

(•) Verdadero. ( ) Falso.

De esta forma podremos referenciar a cualquier objeto del documento sin tener ningún tipo de conflicto.

2.1.- Objeto input de tipo texto.

Cada uno de los 4 elementos de tipo texto de los formularios: **`text`**, **`password`**, **`hidden`** y elementos **`textarea`**, son un elemento dentro de la jerarquía de objetos. Todos los elementos, excepto los tipos **`hidden`**, se mostrarán en la página, permitiendo a los usuarios introducir texto y seleccionar opciones.

Para poder usar estos objetos dentro de nuestros scripts de JavaScript, simplemente será suficiente con asignar un atributo **`id`**, a cada uno de los elementos. Te recomiendo que asignes a cada objeto de tu formulario un atributo **`id`** único y que coincida con el **`name`** de ese objeto.

Cuando se envían los datos de un formulario a un programa en el lado del servidor, lo que en realidad se envía son los atributos **`name`**, junto con los valores (contenido del atributo **`value`**) de cada elemento. Sin lugar a dudas, la propiedad más utilizada en un elemento de tipo texto es por lo tanto **`value`**. Un script podrá recuperar y ajustar el contenido de la propiedad **`value`** en cualquier momento. Por cierto, el contenido de un **`value`** es siempre una cadena de texto, y quizás puedas necesitar realizar conversiones numéricas si quieres realizar operaciones matemáticas con esos textos.

En este tipo de objetos, los gestores de eventos (que verás más adelante) se podrán disparar (*cuando ese evento es ejecutado o es lanzado. Por lo tanto lo podemos traducir por ejecutar o lanzar*) de múltiples formas: por ejemplo, cuando pongamos el foco en un campo (situar el cursor dentro de ese campo) o modifiquemos el texto (al introducir el texto y salir del campo).

```
<!DOCTYPE  html>
<html>
   <head>
      <meta  http-equiv="content-type"  content="text/html;charset=utf-8">
      <title>DWEC05 - Propiedad VALUE de un objeto de tipo texto</title>
      <script  type="text/javascript">
         function convertirMayusculas(){
            /*
            En este ejemplo accedemos a la propiedad value de un objeto con id nombre y le
            asignamos su contenido actual pero convertido a mayúsculas con el método
            toUpperCase() del objeto String.
            */
            document.getElementById("nombre").value=document.getElementById("nombre").
               value.toUpperCase();
         }
      </script>
   </head>
   <body>
      <h1>Propiedad VALUE de un objeto INPUT de tipo TEXT</h1>
      <form id="formulario" action="pagina.php">
            <p>
               <label for="nombre">Nombre y Apellidos: </label>
               <input  type="text"  id="nombre"  name="nombre"  value="" size="30"
                  onblur="convertirMayusculas()">
            </p>
            Introduce tu Nombre y Apellidos y haz click fuera del campo.
      </form>
   </body>
</html>
```

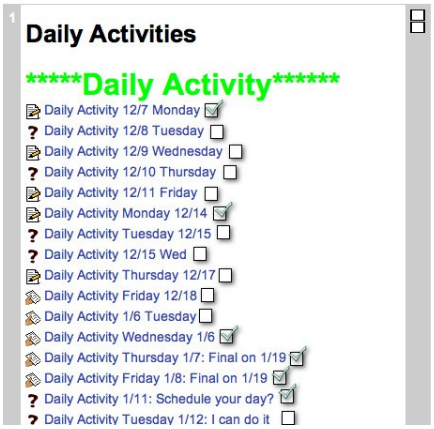
Propiedades del objeto INPUT de tipo texto		
Propiedad	**Descripción**	**W3C**
defaultValue	Ajusta o devuelve el valor por defecto de un campo de texto.	**Sí**
form	Devuelve la referencia al formulario que contiene ese campo de texto.	**Sí**
maxLength	Devuelve o ajusta la longitud máxima de caracteres permitidos en el campo de tipo texto	**Sí**
name	Ajusta o devuelve el valor del atributo `name` de un campo de texto.	**Sí**
readOnly	Ajusta o devuelve si un campo es de sólo lectura, o no.	**Sí**
size	Ajusta o devuelve el ancho de un campo de texto (en caracteres).	**Sí**
type	Devuelve el tipo de un campo de texto.	**Sí**
value	Ajusta o devuelve el contenido del atributo `value` de un campo de texto.	**Sí**

Métodos del objeto INPUT de tipo texto		
Metodo	**Descripción**	**W3C**
select()	Selecciona el contenido de un campo de texto.	Sí

Además de los métodos anteriores, los campos de tipo texto también soportan todas las propiedades estándar, métodos y eventos.

http://www.w3schools.com/jsref/dom_obj_all.asp
http://www.htmlquick.com/es/reference/tags/input.html

2.2.- Objeto input de tipo checkbox.



Un **`checkbox`** es también un objeto muy utilizado en los formularios, pero algunas de sus propiedades puede que no sean muy intuitivas. En los botones de un formulario la propiedad **`value`** nos mostrará el texto del botón, pero en un **`checkbox`** la propiedad **`value`** es un texto que está asociado al objeto. Este texto no se mostrará en la página, y su finalidad es la de asociar un valor con la opción actualmente seleccionada. Dicho valor será el que se enviará, cuando enviemos el formulario.

Por ejemplo:

```
<label  for="cantidad">Si desea recibir 20 Kg marque esta opción: </label>
<input  type="checkbox"  id="cantidad"  name="cantidad"  value="20 Kg">
```

Si chequeamos este **`checkbox`** y enviamos el formulario, el navegador enviará el par **`name/value`** "`cantidad`" y "`20 Kg`". Si el **`checkbox`** no está marcado, entonces este campo no será enviado en el formulario. El texto del **`label`** se mostrará en la pantalla pero las etiquetas **`label`** no se envían al servidor. Para saber si un campo de tipo **`checkbox`** está o no marcado, disponemos de la propiedad **`checked`**. Esta propiedad contiene un valor booleano: `true` si el campo está marcado o `false` si no está marcado. Con esta propiedad es realmente sencillo comprobar o ajustar la marca en un campo de tipo **`checkbox`**.

Veamos el siguiente ejemplo:

```
<html>
   <head>
      <script type="text/javascript">
         function marcar(){
            document.getElementById("verano").checked=true;
         }
         function desmarcar(){
            document.getElementById("verano").checked=false;
         }
      </script>
   </head>
   <body>
      <form action="" method="get">
         <input type="checkbox" id="verano" name="verano" value="Si"/>¿Te gusta el verano?
         <input type="submit" />
      </form>
      <button onclick="marcar()">Marcar Checkbox</button>
      <button onclick="desmarcar()">Desmarcar Checkbox</button>
   </body>
</html>
```

Propiedades del objeto INPUT de tipo checkbox

Propiedad	Descripción	W3C
checked	Ajusta o devuelve el estado `checked` de un `checkbox`.	**Sí**
defaultChecked	Devuelve el valor por defecto del atributo `checked`.	**Sí**
form	Devuelve la referencia al formulario que contiene ese campo `checkbox`.	**Sí**
name	Ajusta o devuelve el valor del atributo `name` de un `checkbox`.	**Sí**
type	Nos indica que tipo de elemento de formulario es un `checkbox`.	**Sí**
value	Ajusta o devuelve el valor del atributo `value` de un `checkbox`.	**Sí**

2.3.- Objeto input de tipo radio.

El ajustar un grupo de objetos de tipo radio desde JavaScript requiere un poco más de trabajo. Para dejar que el navegador gestione un grupo de objetos de tipo radio, deberemos asignar el mismo atributo **`name`** a cada uno de los botones del grupo. Podemos tener múltiples grupos de botones de tipo radio en un formulario, pero cada miembro de cada grupo tendrá que tener el mismo atributo **`name`** que el resto de compañeros del grupo.

Cuando le asignamos el mismo **`name`** a varios elementos en un formulario, el navegador lo que hace es crear un array con la lista de esos objetos que tienen el mismo **`name`**. El contenido del atributo **`name`** será el nombre del array. Algunas propiedades, se las podremos aplicar al grupo como un todo; otras en cambio, tendremos que aplicárselas a cada elemento del grupo y lo haremos a través del índice del array del grupo. Por ejemplo, podemos ver cuántos botones hay en un grupo **`radio`**, consultando la propiedad **`length`** de ese grupo:

```
objetoFormulario.nombregrupo.length
```

Y si queremos acceder a la propiedad **`checked`** de un botón en particular, lo haremos accediendo a la posición del array y a la propiedad **`checked`**:

```
objetoFormulario.nombregrupo[0].checked       // Accedemos a la propiedad checked del primer botón del grupo
```

Ejemplo:

```
<!DOCTYPE HTML>
<html>
   <head>
      <meta charset="utf-8">
      <title>DWEC05 - Trabajando con objetos input de tipo radio</title>
      <script type="text/javascript">
         function  mostrarDatos(){
            for  (var i=0;i<document.formulario.actores.length; i++){
               if  (document.formulario.actores[i].checked)
                  alert(document.formulario.actores[i].value);
            }
         }
      </script>
   </head>
   <body>
      <h1>Trabajando con objetos input de tipo radio</h1>
      <form  name="formulario" action="stooges.php">
         <fieldset>
            <legend>Selecciona tu actor favorito:</legend>
            <input  type="radio"  name="actores"  id="actor-1" value="Walter Bruce Willis - 19 de Marzo de 1955"  checked>
            <label  for="actor-1">Willis</label>

            <input  type="radio"  name="actores"  id="actor-2" value="James Eugene Jim Carrey - 17 de Enero de 1962">
            <label  for="actor-2">Carrey</label>
            <input   type="radio"   name="actores"   id="actor-3" value="Luis Tosar - 13 de Octubre de 1971">
            <label  for="actor-3">Tosar</label>
            <input   type="button"   id="consultar"   name="consultar" value="Consultar Más Datos" onclick="mostrarDatos()">
         </fieldset>
      </form>
   </body>
</html>
```

http://www.desarrolloweb.com/articulos/1448.php

2.4.- Objeto select.

Uno de los controles más complejos que te puedes encontrar en formularios, es el objeto `select`. Un objeto `select` está compuesto realmente de un array de objetos `option`. El objeto `select` se suele mostrar como una lista desplegable en la que seleccionas una de las opciones, aunque también tienes la opción de selecciones múltiples, según definas el objeto en tu documento. Vamos a ver cómo gestionar una lista que permita solamente selecciones sencillas.

Algunas propiedades pertenecen al objeto `select` al completo, mientras que otras, por ejemplo, sólo se pueden aplicar a las opciones individuales dentro de ese objeto. Si lo que quieres hacer es detectar la opción seleccionada por el usuario, y quieres usar JavaScript, tendrás que utilizar propiedades tanto de `select`, como de `option`.

La propiedad más importante del objeto `select` es la propiedad `selectedIndex`, a la que puedes acceder de las siguientes formas:

```
objetoFormulario.nombreCampoSelect.selectedIndex
document.getElementById("objetoSelect").selectedIndex
```

El valor devuelto por esta propiedad, es el índice de la opción actualmente seleccionada, y por supuesto recordarte que los índices comienzan en la posición 0. Siempre que queramos acceder a la opción actualmente seleccionada, recuerda que tendrás que usar esta propiedad.

Las opciones tienen dos propiedades accesibles que son `text` y `value`, y que te permitirán acceder al texto visible en la selección y a su valor interno para esa opción (ejemplo: `<option value="OU">Ourense</option>`). Veamos las formas de acceso a esas propiedades:

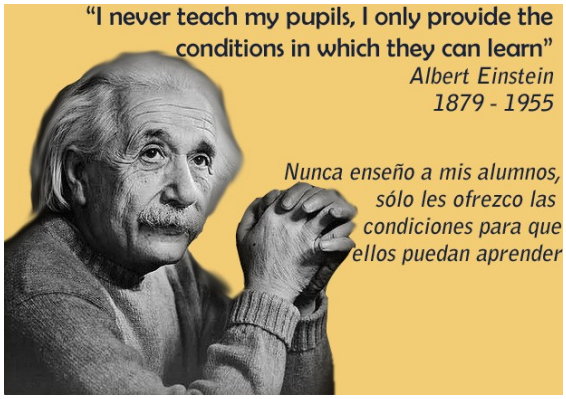
```
objetoFormulario.nombreCampoSelect.options[n].text
objetoFormulario.nombreCampoSelect.options[n].value
```

o también usando `document.getElementById("objetoSelect").options[n].text` **ó** `.value`

```
<!DOCTYPE  html>
<html>
   <head>
      <meta  http-equiv="content-type"  content="text/html;charset=utf-8">
      <title>DWEC05 - Trabajando con un objeto Select</title>
      <script  type="text/javascript">
         function consultar(){
            var objProvincias=document.getElementById("provincias");
            var texto=objProvincias.options[objProvincias.selectedIndex].text;
            var valor=objProvincias.options[objProvincias.selectedIndex].value;
            alert("Datos de la opción seleccionada:\n\nTexto: "+texto+"\nValor: "+valor);
         }
      </script>
   </head>
   <body>
      <h1>Trabajando con un objeto Select</h1>
      <form id="formulario" action="pagina.php">
         <p>
            <label for="provincias">Seleccione provincia: </label>
            <select name="provincias" id="provincias">
               <option value="C">La Coruña</option>
               <option value="LU">Lugo</option>
               <option value="OU">Ourense</option>
               <option value="PO">Pontevedra</option>
            </select>
         </p>
         Selecciona una opción y pulsa el botón.
         <input  type="button"  name="boton"  value="Consultar  información  de  la  opción"
onclick="consultar()"/>
      </form>
   </body>
</html>
```

http://www.htmlquick.com/es/reference/tags/select.html

2.5.- Pasando objetos a las funciones usando this.



En los ejemplos que hemos visto anteriormente cuando un gestor de eventos (`onclick`, `onblur`,...) llama a una función, esa función se encarga de buscar el objeto sobre el cuál va a trabajar. En JavaScript disponemos de un método que nos permite llamar a una función, pasándole directamente la referencia del objeto, sin tener que usar variables globales o referenciar al objeto al comienzo de cada función.

Para conseguir hacerlo necesitamos usar la palabra reservada `this`, la cual hace referencia siempre al objeto que contiene el código de JavaScript en donde usamos dicha palabra reservada. Por ejemplo, si programamos una función para un botón, que al hacer click haga algo, si dentro de esa función usamos la palabra `this`, entonces estaremos haciendo referencia al objeto en el cuál hemos hecho click, que en este caso será el botón. El uso de `this` nos permite evitar usar variables globales, y el programar scripts más genéricos.

Por ejemplo:

```
<!DOCTYPE  html>
<html>
   <head>
      <meta  http-equiv="content-type"  content="text/html;charset=utf-8">
      <title>Uso de la palabra reservada this</title>
      <script  type="text/javascript">
         function identificar(objeto){
            var atrName=objeto.name;
            var atrId=objeto.id;
            var atrValue=objeto.value;
            var atrType=objeto.type;
            alert("Datos  del  campo  pulsado:\n\nName:  "+atrName+"\nID:  "+atrId+"\nValue:
"+atrValue+"\nType: "+atrType);
         }
      </script>
   </head>
   <body>
      <h1>Trabajando con this</h1>
      <form id="formulario" action="pagina.php">
         <p>
            <label for="nombre">Nombre: </label>
            <input      type="text"      name="nombre"      id="nombre"      value="Constantino"
onclick="identificar(this)"/>
            <label for="apellidos">Apellidos: </label>
            <input    type="text"    name="apellidos"    id="apellidos"    value="Veiga    Perez"
onclick="identificar(this)"/>
            <label for="edad">Edad: </label>
            <input        type="password"        name="edad"        id="edad"        value="55"
onclick="identificar(this)"/>
            <label for="pais">Pais: </label>
            España      <input      type="radio"      name="pais"      id="pais1"      value="ES"
onclick="identificar(this)"/>
            Francia      <input      type="radio"      name="pais"      id="pais2"      value="FR"
onclick="identificar(this)"/>
         </p>
         Haga click en cada uno de los campos para ver más información.
      </form>
   </body>
</html>
```

En este ejemplo, cada vez que hagamos click en alguno de los objetos, llamaremos a la función `identificar()` y a esa función le pasaremos como parámetro `this`, que en este caso será la referencia al objeto en el cuál hemos hecho click. La función `identificar()` recibe ese parámetro, y lo almacena en la variable objeto, la cuál le permite imprimir todas las referencias al `name`, `id`, `value`

y `type`. En el siguiente apartado veremos los eventos y allí te mostraré otro uso de `this` por ejemplo dentro de la función `identificar()` sin tener que pasar ningún parámetro.

"Nunca consideres el estudio como un deber, sino como una oportunidad para penetrar en el maravilloso mundo del saber."

EINSTEIN, Albert

3.- Eventos.

Caso práctico
La mayor parte de las veces que un usuario realiza acciones en un formulario está generando eventos. Por ejemplo, cuando hace click con el ratón, cuando sitúa el cursor en un campo, cuando mueve el ratón sobre algún objeto, etc.
Con JavaScript podremos programar que, cuando se produzca alguno de esos eventos, realice una tarea determinada. Es lo que se conoce en programación como ***capturar un evento****.*
Los modelos de registro de esos eventos, así como el orden en el que esos eventos se generan, es la parte que va a estudiar ***Antonio*** *ahora mismo. Esta parte le ayudará mucho cuando tenga que capturar eventos que se produzcan en secuencia, o cuando quiera cancelar un evento, etc.*

Hay que tener en cuenta que, sin eventos prácticamente no hay scripts. En casi todas las páginas web que incorporan JavaScript, suele haber eventos programados que disparan la ejecución de dichos scripts. La razón es muy simple, JavaScript fue diseñado para añadir interactividad a las páginas: el usuario realiza algo y la página reacciona.

Por lo tanto, JavaScript necesita detectar de alguna forma las acciones del usuario para saber cuándo reaccionar. También necesita saber las funciones, que queremos que ejecute cuando se produzcan esas acciones.

Cuando el usuario hace algo se produce un evento. También habrá algunos eventos que no están relacionados directamente con acciones de usuario: por ejemplo el evento de carga (`load`) de un documento, que se producirá automáticamente cuando un documento ha sido cargado.

Todo el tema de gestión de eventos se popularizó a raíz de la versión 2 de Netscape, que también soportaba eventos. Netscape 2 soportaba solamente algunos eventos. Los eventos `mouseOver` y `mouseOut`, se hicieron muy famosos a raíz de su utilización para hacer el efecto de sustitución de una imagen por otra al pasar el ratón por encima. Todo el resto de navegadores, incluido Explorer, tuvieron que adaptarse a la forma en la que Netscape 2 y 3 gestionaban los eventos.

Aunque hoy en día la técnica de gestión de eventos varía con el objetivo de independizar el código de JavaScript de la estructura HTML, los navegadores todavía son compatibles con las técnicas utilizadas por Netscape.

Anteriormente, a las versiones 4 de los navegadores, los eventos (interacciones del usuario y del sistema), eran capturados preferentemente por gestores de eventos definidos como atributos en las etiquetas HTML (modelo de eventos en línea). Por ejemplo, cuando un usuario hacía click en un botón, el evento de `onclick` que se había programado en la etiqueta HTML era disparado. Ese evento hacía una llamada a una función en la que se realizarían las operaciones programadas por el usuario.

Aunque todo ese modo de gestión de eventos sigue funcionando, los navegadores modernos ya incorporan un modelo de eventos, que proporciona un montón de información sobre cómo ocurre un evento. Estas propiedades son accesibles a través de JavaScript, permitiendo programar respuestas más inteligentes a las interacciones del usuario con los objetos del documento.

Incompatibilidades entre navegadores

Muchas veces, lo que se hacía en un principio antes de programar cualquier evento, era detectar qué navegador estábamos utilizando, para saber si nuestro navegador soportaba, o no, los métodos y propiedades que queríamos usar. Por ejemplo:

```
if (Netscape) {
   utilizar modelo Netscape
}
else if (Explorer) {
```

```
   utilizar modelo Microsoft
}
```

Pero hoy en día ni ésto ni el modelo de detección basado en DHTML se recomiendan debido a las diferencias que hay entre todos los navegadores actuales. Por lo tanto hay que intentar usar modelos de detección de eventos estándar y que sean los navegadores los que tengan que adaptarse a ese modelo.

3.1.- Modelo de registro de eventos en línea.

En el modelo de registro de eventos en línea (estandarizado por Netscape), el evento es añadido como un atributo más a la etiqueta HTML, como por ejemplo:

```
<A href="pagina.html" onClick="alert('Has pulsado en el enlace')">Pulsa aqui</a>
```

Cuando hacemos click en el enlace, se llama al gestor de eventos `onClick` (al hacer click) y se ejecuta el script; que contiene en este caso una alerta de JavaScript. También se podría realizar lo mismo pero llamando a una función:

```
<A href="pagina.html" onClick="alertar()">Pulsa aqui</a>
function alertar(){
     alert("Has pulsado en el enlace");
}
```

La mezcla de minúsculas y mayúsculas en los nombres de evento (`onClick`, `onMouseOver`) es sólo por tradición, ya que HTML es insensible a mayúsculas y minúsculas. En cambio en XHTML, sí que los atributos tendrán que ir obligatoriamente siempre en minúsculas: tienes que escribir `onclick` y `onmouseover`. Es muy importante que te fijes en esto, ya que probablemente trabajarás con XHTML y deberás cumplir el estándar: "**todos los nombres de atributos irán siempre en minúscula**".

NO USES EL MODELO DE REGISTRO DE EVENTOS EN LÍNEA

Este modelo no se recomienda, y aunque lo has visto en ejemplos que hemos utilizado hasta ahora, tiene el problema de que estamos mezclando la estructura de la página web con la programación de la misma, y lo que se intenta hoy en día es separar la programación en JavaScript, de la estructura HTML, por lo que este modelo no nos sirve.

En el ejemplo anterior, cuando haces click en el enlace se mostrará la alerta y a continuación te conectará con la pagina.html. En ese momento desaparecerán de memoria los objetos que estaban en un principio, cuando se programó el evento. Esto puede ser un problema, ya que si por ejemplo la función a la que llamamos, cuando se produce el evento, tiene que realizar varias tareas, éstas tendrían que ser hechas antes de que nos conecte con la nueva página.

Éste modo de funcionamiento ha sido un principio muy importante en la gestión de eventos. Si un evento genera la ejecución de un script y además también se genera la acción por defecto para ese objeto entonces:

1. El script se ejecutará primero.
2. La acción por defecto se ejecutará después.

EVITAR LA ACCIÓN POR DEFECTO

A veces es interesante el bloquear o evitar que se ejecute la acción por defecto. Por ejemplo, en nuestro caso anterior podríamos evitar que nos conecte con la nueva pagina.html. Cuando programamos un gestor de eventos, ese gestor podrá devolver un valor booleano `true` o `false`. Eso tendremos que programarlo con la instrucción `return true|false`. `False` quiere decir "no ejecutes la acción por defecto". Por lo tanto nuestro ejemplo quedará del siguiente modo:

```
<A href="pagina.html" onClick="alertar(); return false">Pulsa aqui</a>
```

De esa forma, cada vez que pulsemos en el enlace realizará la llamada a la función `alertar()` y cuando termine ejecutará la instrucción "`return false`", que le indicará al navegador que no ejecute la acción por defecto asignada a ese objeto (en este caso la acción por defecto de un hiperenlace es conectarnos con la página `href` de destino).

También sería posible que nos preguntara si queremos o no queremos ir a la pagina.html. Eso podríamos hacerlo sustituyendo `true` o `false` por una función que devuelva `true` o `false` según la respuesta que demos al "`confirm`":

```
<A href="pagina.html" onClick="return preguntar()">Pulsa aqui</a>

function preguntar(){
     return confirm("¿Desea realmente ir a esa dirección?");
}
```

3.2.- Modelo de registro de eventos tradicional.

En los navegadores antiguos, el modelo que se utilizaba era el modelo en línea. Con la llegada de DHTML, el modelo se extendió para ser más flexible. En este nuevo modelo el evento pasa a ser una propiedad del elemento, así que por ejemplo los navegadores modernos ya aceptan el siguiente código de JavaScript:

```
elemento.onclick = hacerAlgo;  // cuando el usuario haga click en el objeto, se llamará a la función hacerAlgo()
```

Esta forma de registro, no fue estandarizada por el W3C, pero debido a que fue ampliamente utilizada por Netscape y Microsoft, todavía es válida hoy en día. La ventaja de este modelo es que podremos asignar un evento a un objeto desde JavaScript, con lo que ya estamos separando el código de la estructura. Fíjate que aquí los nombres de los eventos sí que van siempre en minúsculas.

```
elemento.onclick = hacerAlgo;
```

Para eliminar un gestor de eventos de un elemento u objeto, le asignaremos `null`:

```
elemento.onclick = null;
```

Otra gran ventaja es que, como el gestor de eventos es una función, podremos realizar una llamada directa a ese gestor, con lo que estamos disparando el evento de forma manual. Por ejemplo:

```
elemento.onclick();
// Al hacer ésto estamos disparando el evento click de forma manual y se ejecutará la función hacerAlgo()
```

SIN PARÉNTESIS

Fíjate que en el registro del evento no usamos paréntesis (). El método `onclick` espera que se le asigne una función completa. Si haces: `element.onclick = hacerAlgo()`; la función será ejecutada y el resultado que devuelve esa función será asignado a `onclick`. Pero ésto no es lo que queremos que haga, queremos que se ejecute la función cuando se dispare el evento.

USO DE LA PALABRA RESERVADA THIS

En el modelo en línea podemos utilizar la palabra reservada `this` cuando programamos el gestor de eventos. Por ejemplo:

```
<A href="pagina.html" ID="mienlace" onClick="alertar(this)">Pulsa aqui</a>
<script type="text/javascript">
   function alertar(objeto){
      alert("Te conectaremos con la página: "+objeto.href);
   }
```

```
</script>

Su equivalente en el <strong>modelo tradicional</strong> sería:
<A id="mienlace" href="pagina.html">Pulsa aqui</a>

<script type="text/javascript">
   document.getElementById("mienlace").onclick = alertar;

   function alertar(){
      alert("Te conectaremos con la página: "+this.href);
   }
</script>
```

Fíjate que estamos usando `this` dentro de la función, sin pasar ningún objeto (tal y como hacíamos en el modelo en línea). En el modelo tradicional el `this` que está dentro de la función, hace referencia al objeto donde hemos programado el evento. También hay que destacar que en este modelo es importante que el hiperenlace sea declarado antes de programar la asignación `onclick`, ya que si por ejemplo escribimos el hiperenlace después del bloque de código de JavaScript, éste no conocerá todavía el objeto y no le podrá asignar el evento de `onclick`. Esto también se podría solucionar, programando la asignación de eventos a los objetos, en una función que se ejecute cuando el documento esté completamente cargado. Por ejemplo con `window.onload=asignarEventos`.

Si por ejemplo queremos que cuando se produzca el evento se realicen llamadas a más de una función lo podremos hacer de la siguiente forma:

```
elemento.onclick = function {llamada1; llamada2 };
```

3.3.- Modelo de registro avanzado de eventos según W3C.

El W3C en la especificación del DOM de nivel 2, pone especial atención en los problemas del modelo tradicional de registro de eventos. En este caso ofrece una manera sencilla de registrar los eventos que queramos, sobre un objeto determinado.

La clave para poder hacer todo eso está en el método `addEventListener()`.

Este método tiene tres **argumentos**: el **tipo de evento**, la **función a ejecutar** y un **valor** booleano (`true` o `false`), que se utiliza para indicar cuándo se debe capturar el evento: en la fase de captura (`true`) o de burbujeo (`false`). En el apartado 3.5 veremos en detalle la diferencia entre estas dos fases.

```
elemento.addEventListener('evento', función, false|true)
```

Por ejemplo para registrar la función `alertar()` de los ejemplos anteriores, haríamos:

```
document.getElementById("mienlace").addEventListener('click',alertar,false);

function alertar(){
     alert("Te conectaremos con la página: "+this.href);
}
```

La ventaja de este método, es que podemos añadir tantos eventos como queramos. Por ejemplo:

```
document.getElementById("mienlace").addEventListener('click',alertar,false);
document.getElementById("mienlace").addEventListener('click',avisar,false);
document.getElementById("mienlace").addEventListener('click',chequear,false);
```

Por lo tanto, cuando hagamos click en "mienlace" se disparará la llamada a las tres funciones. Por cierto, el W3C no indica el orden de disparo, por lo que no sabemos cuál de las tres funciones se ejecutará primero. **Fíjate también, que el nombre de los eventos al usar `addEventListener` no lleva `'on'` al comienzo**.

También se pueden usar funciones anónimas (sin nombre de función), con el modelo W3C:

```
element.addEventListener('click', function () {
     this.style.backgroundColor = '#cc0000';
```

```
}, false)
```

Uso de la palabra reservada this

La palabra reservada `this`, tiene exactamente la misma funcionalidad que hemos visto en el modelo tradicional.

¿QUÉ EVENTOS HAN SIDO REGISTRADOS?

Uno de los problemas de la implementación del modelo de registro del W3C, es que no podemos saber con antelación, los eventos que hemos registrado a un elemento.

En el modelo tradicional si hacemos: `alert(elemento.onclick)`, nos devuelve `undefined`, si no hay funciones registradas para ese evento, o bien el nombre de la función que hemos registrado para ese evento. Pero en este modelo no podemos hacer eso.

El **W3C** en el reciente **nivel 3 del DOM**, introdujo un método llamado `eventListenerList`, que almacena una lista de las funciones que han sido registradas a un elemento. Tienes que tener cuidado con este método, porque no está soportado por todos los navegadores.

Para eliminar un evento de un elemento, usaremos el método `removeEventListener()`:

```
elemento.removeEventListener('evento', función, false|true);
```

Para cancelar un evento, este modelo nos proporciona el método `preventDefault()`.

3.4.- Modelo de registro de eventos según Microsoft.

Microsoft también ha desarrollado un modelo de registro de eventos. Es similar al utilizado por el W3C, pero tiene algunas modificaciones importantes.

Para **registrar un evento**, tenemos que enlazarlo con `attachEvent()`:

```
elemento.attachEvent('onclick', hacerAlgo);
```

o si necesitas dos gestores del mismo evento:

```
elemento.attachEvent('onclick', hacerUnaCosa);
elemento.attachEvent('onclick', hacerOtraCosa);
```

Para **eliminar un evento**, se hará con `detachEvent()`:

```
elemento.detachEvent('onclick', hacerAlgo);
```

Comparando este modelo con el del W3C tenemos dos diferencias importantes:

- ✓ Los eventos siempre burbujean, no hay forma de captura.
- ✓ La función que gestiona el evento está referenciada, no copiada, con lo que la palabra reservada `this` siempre hace referencia a window y será completamente inútil.

Como resultado de estas dos debilidades, cuando un evento burbujea hacia arriba es imposible conocer cuál es el elemento HTML que gestionó ese evento.

Listado de atributos de eventos (**IE**: Internet Explorer, **F**: Firefox, **O**: Opera, **W3C**: W3C Standard.)

Listado de atributos de eventos					
Atributo	**El evento se produce cuando...**	**IE**	**F**	**O**	**W3C**
onblur	Un elemento pierde el foco.	3	1	9	**Sí**
onchange	El contenido de un campo cambia.	3	1	9	**Sí**
onclick	Se hace click con el ratón en un objeto.	3	1	9	**Sí**
ondblclick	Se hace doble click con el ratón sobre un objeto.	4	1	9	**Sí**

onerror	Ocurre algún error cargando un documento o una imagen.	4	1	9	**Sí**
onfocus	Un elemento tiene el foco.	3	1	9	**Sí**
onkeydown	Se presiona una tecla del teclado.	3	1	No	**Sí**
onkeypress	Se presiona una tecla o se mantiene presionada.	3	1	9	**Sí**
onkeyup	Cuando soltamos una tecla.	3	1	9	**Sí**
onload	Una página o imagen terminaron de cargarse.	3	1	9	**Sí**
onmousedown	Se presiona un botón del ratón.	4	1	9	**Sí**
onmousemove	Se mueve el ratón.	3	1	9	**Sí**
onmouseout	Movemos el ratón fuera de un elemento.	4	1	9	**Sí**
onmouseover	El ratón se mueve sobre un elemento.	3	1	9	**Sí**
onmouseup	Se libera un botón del ratón.	4	1	9	**Sí**
onresize	Se redimensiona una ventana o frame.	4	1	9	**Sí**
onselect	Se selecciona un texto.	3	1	9	**Sí**
onunload	El usuario abandona una página.	3	1	9	**Sí**

Más información sobre eventos de teclado, ratón y otros eventos
http://www.w3schools.com/jsref/dom_obj_event.asp

3.5.- Orden de disparo de los eventos.

Imagina que tenemos un elemento contenido dentro de otro elemento, y que tenemos programado el mismo tipo de evento para los dos (por ejemplo el evento click). ¿Cuál de ellos se disparará primero? Sorprendentemente, esto va a depender del tipo de navegador que tengamos.

El problema es muy simple. Imagina que tenemos el siguiente gráfico:

y ambos tienen programado el evento de click. Si el usuario hace click en el elemento2, provocará un click en ambos: elemento1 y elemento2. ¿Pero cuál de ellos se disparará primero?, ¿cuál es el orden de los eventos?

Tenemos **dos Modelos propuestos** por Netscape y Microsoft en sus orígenes:

- ✓ Netscape dijo que, el evento en el elemento1 tendrá lugar primero. Es lo que se conoce como **"captura de eventos**".
- ✓ Microsoft mantuvo que, el evento en el elemento2 tendrá precedencia. Es lo que se conoce como "**burbujeo de eventos**".

Los dos modelos son claramente opuestos. Internet Explorer sólo soporta burbujeo (bubbling). Mozilla, Opera 7 y Konqueror soportan los dos modelos. Y las versiones antiguas de Opera e iCab no soportan ninguno.

Modelo W3C

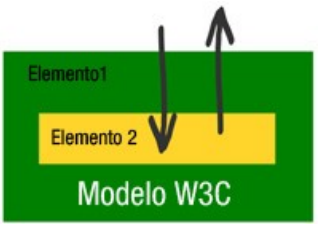
W3C decidió tomar una posición intermedia. Así supone que, cuando se produce un evento en su modelo de eventos, primero se producirá la fase de captura hasta llegar al elemento de destino, y luego se producirá la fase de burbujeo hacia arriba. Este modelo es el estándar, que todos los navegadores deberían seguir para ser compatibles entre sí.

Tú podrás decidir cuando quieres que se registre el evento: en la fase de captura o en la fase de burbujeo. El tercer parámetro de `addEventListener` te permitirá indicar si lo haces en la **fase de captura (`true`)**, o en la **fase de burbujeo(`false`)**.

Por ejemplo:

```
elemento1.addEventListener('click',hacerAlgo1,true);
elemento2.addEventListener('click',hacerAlgo2,false);
```

Si el usuario hace click en el elemento2 ocurrirá lo siguiente:



1. El evento de click comenzará en la fase de captura. El evento comprueba si hay algún ancestro del elemento2 que tenga un evento de `onclick` para la fase de captura (`true`).
2. El evento encuentra un `elemento1.hacerAlgo1()` que ejecutará primero, pues está programado a `true`.
3. El evento viajará hacia el destino, pero no encontrará más eventos para la fase de captura. Entonces el evento pasa a la fase de burbujeo, y ejecuta `hacerAlgo2()`, el cual hemos registrado para la fase de burbujeo (`false`).
4. El evento viaja hacia arriba de nuevo y chequea si algún ancestro tiene programado un evento para la fase de burbujeo. Éste no será el caso, por lo que no hará nada más.

Para **detener la propagación del evento** en la fase de burbujeo, disponemos del método `stopPropagation()`. En la fase de captura es imposible detener la propagación.

4.- Envío y validación de formularios.

Caso práctico
*La validación de un formulario, es una de las típicas tareas que tendrá que desarrollar un programador web. Esta tarea de validación es la que más veces tendrá que realizar **Antonio** en la modernización del portal web, por lo que quiere prestar mucha atención al ejemplo de validación de un formulario con JavaScript, comentado por **Juan**, en el que explica todas las partes del código de validación.*
En ese ejemplo, se utilizan muchas de las referencias que ha visto en esta unidad, y puesto que es un ejemplo básico, es una buena base para una posible ampliación con los requerimientos que se le piden en el proyecto de actualización del portal.

La validación de un formulario es un proceso que consiste en chequear un formulario y comprobar que todos sus datos han sido introducidos correctamente. Por ejemplo, si tu formulario contiene un campo de texto en el que hay que escribir un e-mail, sería interesante comprobar si ese e-mail está escrito correctamente, antes de pasar al siguiente campo.

Hay dos métodos principales de validación de formularios: en el **lado del servidor** (usando scripts CGI, PHP, ASP, etc.) y en el **lado del cliente** (generalmente usando JavaScript).

La validación en el lado del servidor es más segura, pero a veces también es más complicada de programar, mientras que la validación en el lado del cliente es más fácil y más rápida de hacer (el navegador no tiene que conectarse al servidor para validar el formulario, por lo que el usuario recibirá información al instante sobre posibles errores o fallos encontrados).

La idea general que se persigue al validar un formulario, es que cuando se envíen los datos al servidor, éstos vayan correctamente validados, sin ningún campo con valores incorrectos.

A la hora de programar la validación, podremos hacerlo a medida que vamos metiendo datos en el formulario, por ejemplo campo a campo, o cuando se pulse el botón de envío del formulario.

JavaScript añade a tus formularios dos características muy interesantes:
- ✓ JavaScript te permite examinar y validar las entradas de usuario directamente, en el lado del cliente.
- ✓ JavaScript te permite dar mensajes instantáneos, con información de la entrada del usuario.

La validación de datos del usuario en la entrada, generalmente suele fallar en alguna de las 3 siguientes categorías:
- ✓ **Existencia**: comprueba cuando existe o no un valor.
- ✓ **Numérica**: que la información contenga solamente valores numéricos.
- ✓ **Patrones**: comprueba que los datos sigan un determinado patrón, como el formato de un e-mail, una fecha, un número de teléfono, un número de la seguridad social, etc.

JavaScript también se puede utilizar para modificar los elementos de un formulario, basándose en los datos introducidos por el usuario: tal como cubrir un campo de selección con una lista de nombres de ciudades, cuando una determinada provincia está seleccionada, etc.

Una parte muy importante que no debes olvidar al usar JavaScript con formularios, es la posibilidad de que el usuario desactive JavaScript en su navegador, por lo que JavaScript no debería ser una dependencia en la acción de envío de datos desde un formulario.

"ACUÉRDATE DE QUE JAVASCRIPT ESTÁ PARA MEJORAR, NO PARA REEMPLAZAR".

La validación de un formulario en el lado del cliente puede ahorrar algunas idas y vueltas a la hora de enviar los datos, pero aún así, tendrás que realizar la validación de datos en el servidor, puesto que es allí realmente donde se van a almacenar esos datos y el origen de los mismos puede venir por cauces que no hemos programado.

"Cada hombre puede mejorar su vida mejorando su actitud."

TASSINARI, Héctor

4.1.- Ejemplo sencillo de validación de un formulario.

Como te comenté anteriormente el objeto de validar un formulario es comprobar, que antes del envío del mismo, todos los campos poseen valores correctos, evitando así que el usuario tenga que volver de nuevo al formulario, si es que no se pudo validar correctamente en el servidor. Eso no evita que también tengamos que hacer validación en el servidor, ya que recuerda, que el usuario podrá desactivar JavaScript en su navegador, con lo que la validación que hemos programado en JavaScript no funcionaría.

validacionFormulario.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <title>DWEC05 - CONT_R22 Validacion de un Formulario</title>
      <script type="text/javascript" src="validacionFormulario.js"></script>
      <style type="text/css">
         label{
            width: 150px;
            float:left;
            margin-bottom:5px;
         }
         input,select {
            width:150px;
            float:left;
            margin-bottom:5px;
         }
         fieldset{
            background:#CCFF99;
            width:350px;
         }
         .error{
            border: solid 2px #FF0000;
         }
      </style>
   </head>
   <body>
      <fieldset>
         <legend>DWEC05 - Validación de un Formulario -</legend>
         <form name="formulario" id="formulario" action="http://www.google.es" method="get">
            <label for="nombre">Nombre:</label>
            <input type="text" name="nombre" id="nombre" />
            <label for="apellidos">Apellidos:</label>
            <input type="text" name="apellidos" id="apellidos" />
            <label for="edad">Edad:</label>
            <input name="edad" type="text" id="edad" maxlength="3" />
            <label for="matricula">Matrícula Coche:</label>
            <input name="matricula" type="text" id="matricula" value="XXXX AAA" />
            <label for="provincia">Provincia:</label>
            <select name="provincia" id="provincia">
               <option value="0" selected="selected">Seleccione Provincia</option>
               <option value="H">Huesca</option>
               <option value="ZA">Zaragoza</option>
               <option value="T">Teruel</option>
            </select>
            <input type="reset" name="limpiar" id="limpiar" value="Limpiar" />
            <input type="submit" name="enviar" id="enviar" value="Enviar" />
         </form>
      </fieldset>
```

```
   </body>
</html>
```

validacionFormulario.js

```
// Ésta es la primera instrucción que se ejecutará cuando el documento esté cargado.
// Se hará una llamada a la función iniciar(), de esta manera nos aseguramos que las
// asignaciones de eventos no fallarán ya que todos los objetos están disponibles.

window.onload=iniciar;

//-------------------------------------------------------//
function iniciar(){
   // Al hacer click en el botón de enviar tendrá que llamar a la la función validar que
   // se encargará de validar el formulario.
   // El evento de click lo programamos en la fase de burbujeo (false).
   document.getElementById("enviar").addEventListener('click',validar,false);
}

//-------------------------------------------------------//
function validar(eventopordefecto)   {
   // En la variable eventopordefecto gestionaremos el evento por defecto asociado al botón de
   // "enviar" (type=submit) que en este caso lo que hace por defecto es enviar un formulario.
   // Validamos cada uno de los apartados con llamadas a sus funciones correspondientes.
   if    (validarcampostexto(this)    &&    validarMatricula()    &&    validarProvincia()    &&
confirm("¿Deseas enviar el formulario?"))
      return true;
   else
   {
      // Cancelamos el evento de envío por defecto asignado al boton de submit enviar.
      eventopordefecto.preventDefault();
      return false;   // Salimos de la función devolviendo false.
   }
}

//-------------------------------------------------------//
function validarcampostexto(objeto){
   // A esta función le pasamos un objeto (que en este caso es el botón de enviar.
   // Puesto que validarcampostexto(this) hace referencia al objeto dónde se programó ese
   // evento que fue el botón de enviar.
   var formulario = objeto.form;     //  La  propiedad  form  del  botón  enviar  contiene  la
referencia del formulario dónde está ese botón submit.
   for (var i=0; i<formulario.elements.length; i++){
      // Eliminamos la clase Error que estuviera asignada a algún campo.
      formulario.elements[i].className="";
   }

   // De esta manera podemos recorrer todos los elementos del formulario, buscando los que son
   // de tipo texto para validar que contengan valores.
   for (var i=0; i<formulario.elements.length; i++) {
      if (formulario.elements[i].type == "text" && formulario.elements[i].value=="")     {
         alert("El campo: "+formulario.elements[i].name+" no puede estar en blanco");
         formulario.elements[i].className="error";
         formulario.elements[i].focus();
         return false;
      }
      // Aprovechamos y dentro de la función validamos también la edad.
      else if (formulario.elements[i].id=="edad"){
         if   (isNaN(formulario.elements[i].value)   ||   formulario.elements[i].value   <0   ||
formulario.elements[i].value >105)   {
            alert("El  campo:  "+formulario.elements[i].name+"  posee  valores  incorrectos  o  la
edad <0 o >105");
            formulario.elements[i].className="error";
            formulario.elements[i].focus();
            return false;
         }
      }
   }

   return true;    // Si sale de la función con esta instrucción es que todos los campos de
texto y la edad son válidos.
}

//-------------------------------------------------------//
function validarMatricula(){
   // 4 Números 1 espacio en blanco(opcional) y 3 letras de la A-Z en mayúsculas
   var patron = /^\d{4}\s?[A-Z]{3}$/;
```

```
   if (patron.test(document.getElementById("matricula").value))    {
      document.getElementById("matricula").className="";
      return true;
   }else{
      alert("El campo: Matricula no está correcto.\n\nCuatro números espacion en blanco
opcional y 3 letras mayúsculas.");
      // Situamos el foco en el campo matrícula y le asignamos la clase error.
      document.getElementById("matricula").focus();
      document.getElementById("matricula").className="error";
      return false;
   }
}

//-------------------------------------------------------------//
function validarProvincia(){
   // Comprueba que la opción seleccionada sea diferente a 0.
   // Si es la 0 es que no ha seleccionado ningún nombre de Provincia.
   if (document.getElementById("provincia").selectedIndex==0)      {
      alert("Atención!: Debes seleccionar una provincia.");
      // Situamos el foco en el campo provincia y le asignamos la clase error.
      document.getElementById("provincia").focus();
      document.getElementById("provincia").className="error";
      return false;
   }else
      return true;
}
```

Te recomiendo que también veas el vídeo en el que se realiza la validación de un formulario, siguiendo el modelo de registro de eventos en línea. Aunque este modelo se utiliza cada vez menos, todavía te encontrarás muchísimos ejemplos que hacen uso del mismo.

En el siguiente enlace puedes ver un video práctico en el que se muestra un ejemplo de validación de formulario con JavaScript empleando el modelo de registro de eventos en línea:

http://www.youtube.com/watch?feature=player_embedded&v=_FOMVkvdNhQ

Resumen del vídeo:

*Se muestra como crear un formulario dentro de una tabla. Este formulario contiene un campo de tipo texto con el atributo `name="user"`, otro campo de tipo `password` con el atributo `id="pass"` y un botón de tipo `submit`. A continuación se abre código de JavaScript en la parte superior y se crea una función `verify()`. Dentro de la función se definen tres variables `user`, `pass` y `msg`. A la variable `user` le asigna el valor del campo `user` con `user=document.getElementById("user")`, a la variable `pass` le asigna el valor del campo `pass` con el código de JavaScript `var pass=document.getElementById("pass")` y a la variable `msg` le asigna `msg=""`. A continuación comprueba si `user.value` es igual a vacío y si se cumple la condición a la variable msg le concatena un mensaje de error indicando que por favor introduzca un nombre de usuario y después le asigna una clase de CSS al campo `user` llamada `inpBoxError` con la instrucción `user.className="inpBoxError"`. Realiza las mismas acciones con el campo `password`. Después de realizar esas dos comprobaciones chequea si la variable msg es vacío devuelve `true` con `return true`; en otro caso muestra una alerta con la variable `msg` y devuelve `false`. Antes del código de JavaScript abre código de CSS y crea dos clases, una llamada `inpBox` donde define un color de fondo y un borde para los campos de tipo input, y otra clase llama `inpBoxError` donde define un borde de color rojo y ancho de 2 px. Esta clase es la que nos marcará con borde rojo aquellos campos del formulario que contengan errores. A los campos `user` y `pass` les aplica la clase `inpBox` en el código HTML empleando el atributo `class="inpBox"`. Y por último para que se produzca la validación en el momento de envío del formulario programa el evento en línea en la marca **form** con la siguiente instrucción: `onSubmit="return verify()"`. Cuando se produzca el evento de `submit` del formulario se llamará a la función `verify()` que devolverá `true` o `false` dependiendo de si la validación ha sido o no correcta, lo cual provocará que el evento se permita o se cancele.*

¿Cuáles de los siguientes métodos son, los que se utilizan en el modelo de registro de eventos de la W3C?

- [x] addEventListener()
- [ ] onClick()
- [x] removeEventListener()
- [ ] attachEvent()
- [ ] detachEvent()

Sólo addEventListener() y removeEventListener() son métodos utilizados en el registro de eventos de W3C. OnClick() no sería válido por ser un atributo de evento y además tendría que ir en minúsculas y las opciones attachEvent y detachEvent pertenecen al modelo de registro de eventos de Microsoft.

5.- Expresiones regulares y objetos RegExp.

Caso práctico
Antonio *ha comenzado a trabajar en un pequeño formulario del proyecto, y se da cuenta de que en uno de los campos tiene que solicitar al cliente que introduzca los datos en un formato determinado: los datos irán en grupos de caracteres separados por guiones y algunos datos tendrán que llevar paréntesis.*
Antonio *se pone en contacto con* ***Juan****, y le pregunta cómo puede hacer para validar que los datos sigan un formato o estructura en un campo de texto.* ***Juan*** *le indica que hay dos formas principales: una lenta y una rápida.*
La forma lenta, consiste en trabajar con esa cadena de texto, y con los métodos del objeto ***String*** *comprobar que se cumple el formato solicitado. Éste método implicará bastante lógica de programación por lo que dependiendo de la complejidad del formato a chequear habrá que evaluar varias condiciones.*
La forma rápida, y recomendada por ***Juan****, es el uso de expresiones regulares. De esta forma se podría realizar la validación solicitada, por muy complicada que sea, en dos o tres líneas prácticamente, pero eso sí, tendrá que aprender como crear una expresión regular y los caracteres utilizados en expresiones regulares. Pero el esfuerzo merecerá la pena de sobra, ya que el tiempo que le llevará a* ***Antonio****validar un campo empleando el método más lento, será el tiempo que necesitará para aprender expresiones regulares.*

LAS EXPRESIONES REGULARES SON PATRONES DE BÚSQUEDA, QUE SE PUEDEN UTILIZAR PARA ENCONTRAR TEXTO QUE COINCIDA CON EL PATRÓN ESPECIFICADO.

Ejemplo de búsqueda de una cadena de texto sin usar expresiones regulares:

```
var texto = "La linea de alta velocidad llegará pronto a toda España,";
var subcadena = "velocidad";
var i = texto.indexOf(subcadena); // devuelve 17, índice de donde se encuentra la subcadena

if (i != -1) // correcto, se ha encontrado la subcadena
```

Este código funciona porque estamos buscando una subcadena de texto exacta. ¿Pero qué pasaría si hiciéramos una búsqueda más general? Por ejemplo si quisiéramos buscar la cadena "car" en textos como "cartón", "bicarbonato", "practicar", ...?

CUANDO ESTAMOS BUSCANDO CADENAS QUE CUMPLEN UN PATRÓN EN LUGAR DE UNA CADENA EXACTA, NECESITAREMOS USAR EXPRESIONES REGULARES.

Podrías intentar hacerlo con funciones de `String`, pero al final, es mucho más sencillo hacerlo con expresiones regulares, aunque la sintaxis de las mismas es un poco extraña y no necesariamente muy amigable.
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/;
```

La expresión regular está contenida entre la barras `/`, y 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 están hechas de caracteres, solos o en combinación con caracteres especiales, que se proporcionarán para búsquedas más complejas. Por ejemplo, lo siguiente es una expresión regular que realiza una búsqueda que contenga las palabras *Aloe Vera*, en ese orden y separadas por uno o más espacios en medio:

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

Los caracteres especiales en este ejemplo son, la barra invertida (`\`), que tiene dos efectos: o bien se utiliza con un carácter regular, para indicar que se trata de un carácter especial, o se usa con un carácter especial, tales como el signo más (`+`), para indicar que el carácter debe ser tratado

literalmente. En este caso, la barra invertida se utiliza con "`\s`", que transforma la letra `s` en un carácter especial indicando un espacio en blanco, un tabulador, un salto de línea, etc. El símbolo `+` indica que el carácter anterior puede aparecer una o más veces.

5.1.- Caracteres especiales en expresiones regulares.

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

Carácter	Coincidencias	Patrón	Ejemplo de cadena
^	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: "seeee" y también "se".
?	Coincide 0 o 1 vez	**/ap?**	Que la p aparezca 0 o 1 vez: "apple" y "and".
+	Coincide 1 o más veces	**/ap+/**	Que la "p" aparezca 1 o más veces: "apple" pero no "and".
{n}	Coincide exactamente n veces	**/ap{2}/**	Que la "p" aparezca exactamente 2 veces: "apple" pero no "apabullante".
{n,}	Coincide n o más veces	**/ap{2,}/**	Que la "p" aparezca 2 o más veces: "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: "appppppple" (encontrará 4 "p").
.	Cualquier carácter excepto nueva línea	**/a.e/**	Que aparezca cualquier carácter, excepto nueva línea entre la a y la e: "ape" y "axe".
[...]	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".
[^...]	Cualquier carácter excepto los que están entre corchetes	**/a[^px]/**	Que aparezca cualquier carácter excepto la "p" o la "x" después de la letra a: "ale", pero no "axe" o "ape".
\b	Coincide con el inicio de una palabra	**/\bno/**	Que "no" esté al comienzo de una palabra: "novedad".
\B	Coincide al final de una palabra	**/\Bno/**	Que "no" esté al final de una palabra: "este invierno" ("no" de "invierno").
\d	Dígitos del 0 al 9	**/\d{3}/**	Que aparezcan exactamente 3 dígitos: "Ahora en 456".
\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 "Ahora en 456".
\w	Coincide con caracteres del tipo (letras, dígitos, subrayados)	**/\w/**	Que aparezca un carácter (letra, dígito o subrayado): "J" en "JavaScript".
\W	Coincide con caracteres que no sean (letras, dígitos, subrayados)	**/\W/**	Que aparezca un carácter (que no sea letra, dígito o subrayado): "%" en "100%".
\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		
(x)	Capturando paréntesis		Recuerda los caracteres.
\r	Un retorno de carro		
?=n	Cualquier cadena que está seguida por la cadena n indicada después del igual.	**/la(?= mundo)**	Hola mundo mundial.

5.2.- 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");
```

¿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 sea utilizado el modificador "**g**".
ignoreCase	Especifica que sea utilizado el modificador "**i**".
lastIndex	El índice donde comenzar la siguiente búsqueda.
multiline	Especifica si el modificador "**m**" es utilizado.
source	El texto de la expresión regular **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`.

validación.html

```
<html>
   <head>
      <meta charset="utf-8">
      <title>Ejemplo de formulario</title>
      <script type="text/javascript" src="validacion.js"></script>
   </head>
   <body>
      <form name="formulario" id="formulario" method="post" action="">
         <p>
            <label for="nombre">Nombre;</label>
            <input type="text" name="nombre" id="nombre">
         </p>
         <p>
            <label for telefono">Teléfono:</label>
            <input type="text" name="telefono" id="telefono">
         </p>
         <p>
            <label for="matricula">Matrícula coche:</label>
            <input type="text" name="matricula" id="matricula">
         </p>
         <p>
            <input type="reset" name="button" id="button" value="Restablecer">
            <input type="button" name="enviar" id="enviar" value="Enviar">
```

```
            <br />
         </p>
      </form>
   </body>
</html>
```

validación js

```
window.onload=iniciar;

function iniciar(){
   document.getElementById("enviar").addEventListener('click',validar,false);
}

function validar(){
   // Teléfono: 123456789
   var patronTelefono= /^\d{9}$/;

   //Matrícula: 1234 ABC
   var patronMatricula=/^\d{4} [A-Z]{3}$/;

   if (patronTelefono.test(document.getElementById("telefono").value)){
      if(patronMatricula.test(document.getElementById("matricula").value)){
         if(confirm("¿Desea enviar el formulario?")){
            document.getElementById("formulario").submit();
         }
      }else{
         alert("Error: campo matricula incorrecta");
      }
   }else{
      alert("Error: campo telefono incorrecto");
   }
}
```

¿Qué modelo de registro de eventos es el más recomendado?:

- ○ Registro de eventos en línea.
- ○ Registro de eventos tradicional.
- ◉ **Registro de eventos según W3C.**
- ○ Registro de eventos según Microsoft.

Se supone que todos los navegadores que quieran ser compatibles, deberán adaptarse a la normativa del W3C.

5.3.- Ejemplos de uso de expresiones regulares.

Aquí te pongo algunos ejemplos de uso de expresiones regulares:

Comprobando si una subcadena existe dentro de otra

```
<script type="text/javascript">
var datos = new Array();
datos[0] = "El Blogger de Google";          // verdadero
datos[1] = "El blogger de Google";          // falso
datos[2] = "BloggerGoogle";                 // verdadero
datos[3] = "Google Blogger";  // falso

var patron = /Blog.*Goog/;
// Patrón de búsqueda
// Contenga "Blog" cualquier caracter(.) 0 o más veces (*) y a continuación "Goog"
for (var i = 0; i < datos.length; i++)
  alert(datos[i] + " " + patron.test(datos[i]));
</script>
```

Validación de un número de Seguridad Social Americano

Un número de Seguridad Social americano consiste en 8 dígitos, generalmente escritos en tres campos separados por guiones: AAA-GG-SSS. Los tres primeros dígitos se denominan el "número de área". Los dos dígitos centrales son el "número de grupo" y los 3 finales son el "número de serie". En este ejemplo se valida que un número facilitado cumpla el formato indicado con o sin guiones ya que quedan como opcionales (caracteres "-?" dentro de la expresión regular):

```
<!DOCTYPE  html>
```

```
<html>
   <head>
      <meta  http-equiv="content-type"  content="text/html;charset=utf-8">
      <title>Ejemplos de RegExp</title>
      <script type="text/javascript">
         window.onload=iniciar;
         // Cuando se termine de cargar el documento,llamará a la función iniciar().
         // En esa función programaremos el evento de click para el botón comprobar.
         // Si no esperamos a que se cargue el documento, el botón no estará disponible
         // y fallará el acceso a document.getElementByid("comprobar"), ya que todavía no
         // conoce ese objeto.

         function iniciar(){
            document.getElementById("comprobar").onclick=comprobar;
         }

         function comprobar(){
            var numero = document.getElementById("ssn").value;
            var patron = /^\d{3}-?\d{2}-?\d{3}$/;
            if (numero.match(patron))
               alert("Correcto: el número "+numero+" cumple el estándar americano");
            else
               alert("Error: el número "+numero+" NO cumple el estandar.");
         }
      </script>
   </head>
   <body>
      <form name="formulario">
         <label for="ssn">Número Seguridad Social Americano:</label>
         <input type="text" name="ssn" id="ssn" />
         <input type="button" name="comprobar" id="comprobar" value="Comprobar Formato" />
      </form>
   </body>
</html>
```

6.- Las cookies.

Caso práctico
***Antonio**, cuando comenzó a estudiar JavaScript, vio que una de las restricciones que tenía es que no podía escribir ficheros en el ordenador del cliente, a excepción de las **cookies**.*
*Las **cookies** son unos pequeños ficheros, que se escribirán en una carpeta determinada en el ordenador del cliente. Estos ficheros van a permitir a **Antonio** por ejemplo, el grabar cierta información que quiera que permanezca entre diferentes sesiones de trabajo de ese usuario (del mismo día o de días diferentes).*
***Antonio** quiere utilizar las **cookies** para dar un mensaje de aviso al usuario cada vez que entre en la página web, indicándole cuando fue la última vez que visitó la página, y también quiere usar las **cookies** como sistema de carrito de la compra para la tienda que está incluida en la web. De esta forma, podrá almacenar los productos que el usuario va comprando y cuando vaya a realizar el pago, leerá el carrito y mostrará el total a pagar, junto con el listado de productos.*

Permitir que algún programa pueda leer y escribir en el disco duro puede dar que pensar en un principio, pero el mecanismo de las **cookies** es algo más seguro, ya que no abre tu disco duro al mundo para que cualquiera pueda ver su contenido o modificarlo. Este mecanismo de las **cookies** proporciona acceso a un fichero de texto (en Internet Explorer) o a un fichero especial (en otros navegadores distintos a Internet Explorer), que está situado en un directorio especial del disco duro.

En navegadores basados en **Mozilla**, el fichero de cookie se nombra como `cookies.txt` y está localizado en un directorio (cuyo nombre termina en `.slt`) dentro del perfil del navegador.
En **Windows**, esa localización está en:
`C:\\Windows\Application Data\Mozilla\Profiles\[profilename]\;`
y en **Mac OSX** en
`[user]/Library/Mozilla/Profiles/[profilename]/.`

Internet Explorer para Windows usa un sistema diferente: todas las cookies para cada dominio se almacenarán en un fichero específico de dominio dentro del directorio `C:\Windows\Temporary Internet Files\`. Los ficheros comienzan con el nombre de `Cookie`: e incluyen el usuario y dominio del servidor que escribió la cookie.

En **Safari** las cookies son almacenadas en un fichero XML llamado `Cookies.plist`.

Google Chrome almacena las cookies en su base de datos `SQLite` en un fichero llamado `Cookies` dentro de `[user] \ Local Settings \ Application Data \ Google \ Chrome \ User Data \ Default`. Tienes que tener precaución con Chrome ya que este navegador deshabilita las cookies si no accedemos a través de `http://`

Un fichero de cookies es un fichero de texto. El formato de almacenamiento de los datos en ese fichero dependerá del navegador. La estructura de ese fichero te dará igual ya que para acceder a las cookies lo vas a hacer a través de la propiedad `document.cookie`.

Formato de un registro de cookie

Entre todos los campos que se almacenarán en una cookie tenemos los siguientes (no necesariamente en el mismo orden):

- ✓ Dominio del servidor que creó la cookie.
- ✓ Información de si es necesaria una conexión http segura para acceder a la cookie.
- ✓ Trayectoria de las URL que podrán acceder a la cookie.
- ✓ Fecha de caducidad de la cookie.
- ✓ Nombre de una entrada de datos.
- ✓ Cadena de texto asociada a ese nombre de entrada de datos.

LAS COOKIES SON ESPECÍFICAS AL DOMINIO. En otras palabras, si un dominio crea una cookie, otro dominio no podrá acceder a ella a través del navegador. La razón de ello es que muchas veces podremos almacenar datos importantes como usuarios/contraseñas en las cookies, y no queremos que otros dominios puedan consultarlos. La mayor parte de las veces, cuando almacenamos datos de este tipo, estarán encriptados dentro de lacookie.

LAS COOKIES TENDRÁN UNA FECHA DE CADUCIDAD, ya que algunos navegadores tienen limitado el número máximo de cookies que pueden almacenar (1000 en Firefox). Será el propio navegador el encargado de borrar las cookies caducadas.

6.1.- Gestión y uso de cookies.

Grabar una cookie

Para grabar datos en un fichero de cookie, podemos utilizar una asignación simple con la propiedad `document.cookie`. Pero tendrás que tener mucha precaución con el formato de datos, ya que si no la cookie no será grabada correctamente. Aquí te muestro la sintaxis de cómo se asignaría un valor a una cookie (los campos opcionales van entre corchetes; en cursiva irán las posiciones para escribir nuestros propios datos):

```
document.cookie  =  "nombreCookie=datosCookie
          [;  expires=horaformatoGMT]
          [;  path=ruta]
          [;  domain=nombreDominio]
          [;  secure]"
```

Cada cookie deberá tener un nombre y un texto asignado (aunque sea cadena vacía ""). Por ejemplo si quieres almacenar la cadena "**Martin**" para una cookie "**usuario**", haríamos:

```
document.cookie = "usuario=Martin";
```

El navegador ve que no tenemos ninguna cookie con este nombre, la creará automáticamente; si la cookie ya existe, entonces la reemplazará. Se pueden omitir el resto de parámetros de la cookie; en ese caso el navegador usará valores por defecto. Para cookies temporales generalmente sólo necesitaremos escribir nombre/valor. Es decir estas cookies durarán solamente el tiempo de la sesión. Si por ejemplo cerramos el navegador y lo volvemos a abrir la cookie desaparece.
Ejemplo:

```
document.cookie="contador=0";
// Almacena contador=0 en la cookie sin ningún otro contenido a mayores.
```

La fecha de caducidad **expires**, tendrá que ir en formato GMT. Por ejemplo:

```
expires=Thu,  01-Jan-70  00:00:01  GMT;
```

El `path` será la ruta actual de nuestra página web.

El dominio `domain` si no se pone, se asignará por defecto el dominio de la página que creó la cookie.

Si se omite el valor `secure`, ésto implica que nuestra cookie será accesible por cualquier programa en nuestro dominio que se ajuste a las propiedades de dominio y `path`.

Recuperar información de una cookie

Para recuperar los datos de una cookie tendremos que acceder a la propiedad `document.cookie` e imprimir su valor. Los resultados nos llegarán en forma de cadena de texto. Por ejemplo una cadena de texto `document.cookie` podría tener el siguiente aspecto:

```
usuario=Martin; password=OjYgdjUA
```

En otras palabras no podremos tratar las cookies como objetos. En su lugar, deberemos pasar la cadena de texto de la cookie, y extraer los datos necesarios de su contenido.

Cada vez que nuestro ordenador pide una página a un servidor, este realiza una conexión nueva con el servidor, por lo cual, el servidor no tiene conocimiento de las anteriores acciones del visitante (Por ejemplo logins).

Para resolver eso, nació un tipo de archivo, llamado cookie, que se almacena en el ordenador del visitante y puede contener información sobre nuestros movimientos.

Así, una vez entramos a un servicio con nuestro nombre y contraseña, el servidor, nos suele identificar con un número al azar que es guardado en el servidor y enviado a la vez al usuario, de manera que con ese número, puede conocer nuestro nombre, contraseña...

Desde JavaScript, el proceso de escritura y borrado de una cookie es muy sencillo:

```
document.cookie="NOMBRE = VALOR; expires = FECHA"
```

Así por ejemplo podemos guardar una cookie llamada ID con valor 123456 con caducidad el 2 de Diciembre del 2014:

```
document.cookie="ID = 123456; expires = 2 Dec 2014 23:59:59 GMT"
```

Y para borrarla le definimos una fecha de caducidad por ejemplo de 1995.

En cambio para leer una cookie deberemos crear una función especial:

```
function leerCookie(nombre) {
   a = document.cookie.substring(document.cookie.indexOf(nombre + '=') + nombre.length +
1,document.cookie.length);
   if(a.indexOf(';') != -1)a = a.substring(0,a.indexOf(';'))
   return a;
}
```

A la que llamaremos desde:

```
alert(leerCookie('ID')) , document.write(leerCookie('ID'))...
```

¿Te has parado a pensar que pasaría si el usuario bloquea en el navegador las cookies o JavaScript?

TEMA 6

Contenido

Modelo de objetos del documento en javascript.

Caso práctico
Antonio está haciendo muchos avances en su proyecto. Por ejemplo, ya ha realizado casi toda la validación de los formularios de las páginas web empleando JavaScript, y les ha añadido mensajes de errores de forma dinámica. Ahora se encuentra con un pequeño problema: necesita poder acceder a algunas partes de su documento y modificar contenidos, pero no sabe cómo hacerlo. Le interesaría hacer cosas, como borrar celdas en tablas, añadir o modificar atributos a elementos, modificar contenido textual de cualquier parte del documento, etc. y todo ello usando JavaScript.
Juan le informa que todo eso que solicita se puede hacer con JavaScript, pero tendrá que hacer uso del DOM de una forma más intensiva. El DOM organiza todo el documento en una especie de árbol de elementos, de tal forma que, a través de JavaScript, podrá acceder a cada uno de esos nodos y modificar su contenido, añadir nuevos nodos, eliminarlos, recorrerlos, etc. Cualquier cambio que realice en el árbol de nodos, es reflejado de forma automática por el navegador web, con lo que las modificaciones son instantáneas de cara al cliente.
Por último, Antonio pregunta si es posible detectar qué botones del ratón han sido pulsados, o si se está utilizando una combinación de teclas en el teclado, ya que, por ejemplo, una de las cosas que quiere hacer en sus formularios es que, cuando se esté dentro de un campo de texto y se pulse Enter se pase automáticamente al siguiente campo, o que cuando se pulse una combinación de teclas determinada, se realice una tarea que tenga programada en JavaScript.
Juan le responde que para hacer eso tiene que profundizar un poco más en los eventos y, en especial, en el objeto Event, que le permite acceder a nuevas propiedades que le proporcionarán esa información específica que busca. Juan además puntualiza que, ahora que se mete de lleno en eventos más específicos, tendrá que tener en cuenta las incompatibilidades entre navegadores, para que sus aplicaciones sean multi-cliente. Para conseguirlo le da unas indicaciones de cómo tiene que programar los eventos, y qué diferencias va a encontrar entre los distintos navegadores.

1.- Bases del Modelo de Objetos del Documento (DOM).

Caso práctico
*El estudio más a fondo del DOM, va a permitir a **Antonio** llegar a conocer con muchísimo detalle cómo se construye una página web, ya que el DOM es la base de toda la estructura de cualquier documento. El conocer a fondo el DOM, qué tipos de nodos contiene, cómo acceder a ellos para recorrerlos, modificarlos o borrarlos, y ver las diferentes aproximaciones según los navegadores, supondrá un salto cualitativo en su programación con JavaScript. De esta forma, prácticamente cualquier cosa que se proponga dejará de tener secretos, porque prácticamente todo lo que se vea en la página Web, va a estar accesible a través de JavaScript empleando las instrucciones de manejo del DOM.*

El DOM (Modelo de Objetos del Documento), es esencialmente una interfaz de programación de aplicaciones (API), que proporciona un conjunto estándar de objetos, para representar documentos HTML y XML, un modelo estándar sobre cómo pueden combinarse dichos objetos, y una interfaz estándar para acceder a ellos y manipularlos. A través del DOM los programas pueden acceder y modificar el contenido, estructura y estilo de los documentos HTML y XML, que es para lo que se diseñó principalmente. El responsable del DOM es elW3C.

El DOM está separado en 3 partes / niveles:

- ✓ DOM Core modelo estándar para cualquier documento estructurado.
- ✓ DOM XML modelo estándar para documentos XML.
- ✓ DOM HTML modelo estándar para documentos HTML.

EL DOM HTML es un estándar dónde se define cómo acceder, modificar, añadir o borrar elementos HTML.

En el DOM se definen los **objetos** y **propiedades** de todos los elementos del documento, y los métodos para acceder a ellos.

Es importante citar que en el DOM del W3C, no se especifican todas las características especiales de los modelos de objeto de exploración. Muchas de las funciones de Internet Explorer 4 (y posteriores) del modelo de objetos, no forman parte de la especificación DOM W3C.

Debes tener en cuenta que, mientras que los navegadores basados en Mozilla están haciendo grandes esfuerzos para poner en práctica todos los niveles del DOM 1 y la mayoría de Nivel 2 del W3C, Microsoft (por la razón que sea) sólo realiza una aplicación parcial del DOM a sus navegadores, aunque con las versiones más modernas se están adaptando poco a poco al estándar. Otros navegadores modernos como Chrome, Safari, Opera, soportan de forma extensiva el DOM del W3C.

Niveles del DOM
Al igual que otras muchas especificaciones del W3C, una versión no suele ser suficiente, por lo que la especificación del DOM sigue el mismo camino. El DOM está en continua evolución. Las fechas propuestas de las diferentes versiones aportadas por el W3C, raramente coinciden con las versiones de los navegadores. Por lo que suele ser muy común que muchas versiones de los navegadores incluyan solamente, algunos detalles de las versiones más recientes del W3C. La primera especificación DOM nivel 1, fue liberada después de Netscape4 e Internet Explorer 4. La parte de HTML cubierta por la especificación de nivel 1 incluye el llamado DOM de nivel 0 (aunque no hay ningún estándar publicado con ese nombre). Esta especificación es esencialmente el modelo de objetos implementado en Navigator 3 (y en parte de Internet Explorer 3 incluyendo el objeto image). Quizás la parte omitida que podamos destacar de este modelo de nivel 1, ha sido una especificación del modelo de eventos.

El DOM de nivel 2 trabaja sobre los desarrollos de nivel 1. Se han añadido nuevas secciones, estilos, formas de inspección de la jerarquía del documento, y se han publicado como módulos separados. Algunos módulos del nivel 3 del DOM han alcanzado el estado de "Recomendación". Aunque Internet Explorer sigue sin implementar una gran mayoría de opciones de los módulos, otros navegadores sí que implementan algunos de los módulos, incluso de los que están en estado de "Borrador".

Calendario de actividades del DOM en W3C. http://www.w3schools.com/w3c/w3c_dom.asp

1.1.- Objetos del DOM HTML, propiedades y métodos.

En otras unidades ya has trabajado con muchos de los objetos que te presentamos en esta lista. Aquí se muestra la referencia completa de objetos, que puedes encontrar en el Modelo de Objetos del Documento para HTML. Y, al final de la lista, hay un hiperenlace que debes visitar para ampliar información sobre las propiedades y métodos de aquellos objetos que no hayas utilizado hasta este momento.

Te recuerdo aquí la sintaxis para acceder a las propiedades o métodos de aquellos objetos que estén dentro de nuestro documento:

```
document.getElementById(objetoID).propiedad | metodo( [parametros] )
```

NOTA: Los datos entre corchetes son opcionales.

Listado de objetos del DOM en HTML:

Document	HTMLElement	Anchor	Area	Base
Body	Button	Event	Form	Frame/IFrame
Frameset	Image	Input Button	Input Checkbox	Input File
Input Hidden	Input Password	Input Radio	Input Reset	Input Submit
Input Text	Link	Meta	Object	Option
Select	Style	Table	TableCell	TableRow
Textarea				

Introducción al DOM. **http://www.librosweb.es/ajax/capitulo4.html**

"Por muy alto que sea un árbol, sus hojas siempre caen hacia la raíz."

Anónimo

1.2.- El árbol del DOM y tipos de nodos.

La tarea más habitual en programación web suele ser la manipulación de páginas web, para acceder a su contenido, crear nuevos elementos, hacer animaciones, modificar valores, etc.

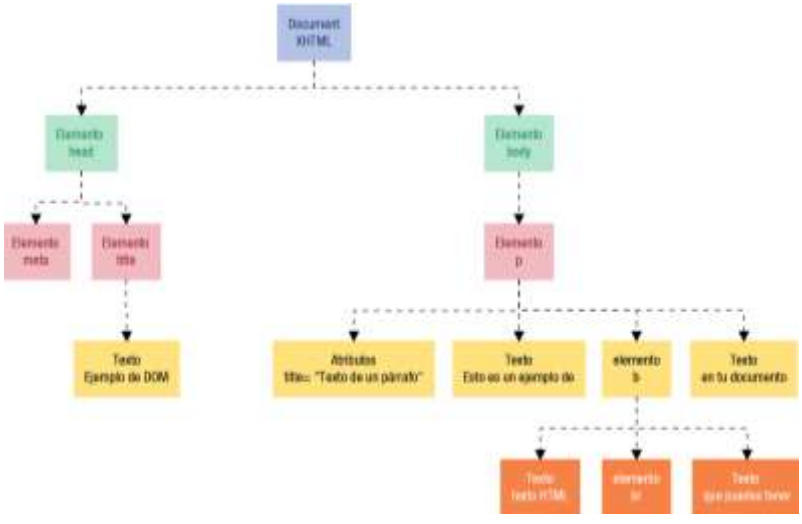
Todas estas tareas se pueden realizar de una forma más sencilla gracias al DOM. Los navegadores web son los encargados de realizar la transformación de nuestro documento, en una estructura jerárquica de objetos, para que podamos acceder con métodos más estructurados a su contenido.

El DOM transforma todos los documentos XHTML en un conjunto de elementos, a los que llama **nodos**. En el HTML DOM **cada nodo es un objeto**. Estos nodos están conectados entre sí y representan los contenidos de la página web, y la relación que hay entre ellos. Cuando unimos todos estos nodos de forma jerárquica, obtenemos una estructura similar a un árbol, por lo que muchas veces se suele referenciar como árbolDOM, "**árbol de nodos**", etc.

Veamos el siguiente ejemplo de código:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1ransitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <title>Ejemplo de DOM</title>
   </head>
   <body>
      <p title="Texto de un párrafo">Esto es un ejemplo de <b>texto HTML<br />que puedes
tener</b> en tu documento.</p>
   </body>
</html>
```

Ese código se transformaría en el siguiente árbol de nodos:



Cada rectángulo del gráfico representa un nodo del DOM, y las líneas indican cómo se relacionan los nodos entre sí. La raíz del árbol de nodos es un nodo especial, denominado "**document**". A partir de ese nodo, cada etiqueta XHTML se transformará en nodos de tipo "**elemento**" o "**texto**". Los nodos de tipo "texto", contendrán el texto encerrado para esa etiqueta XHTML. Esta conversión se realiza de forma jerárquica. El nodo inmediatamente superior será el **nodo padre** y todos los nodos que están por debajo serán **nodos hijos**.

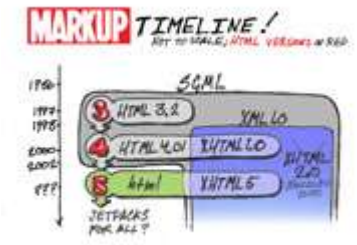
Tipos de nodos.

La especificación del DOM define 12 tipos de nodos, aunque generalmente nosotros emplearemos solamente cuatro o cinco tipos de nodos:

- ✓ `Document`, es el nodo raíz y del que derivan todos los demás nodos del árbol.
- ✓ `Element`, representa cada una de las etiquetas XHTML. Es el único nodo que puede contener atributos y el único del que pueden derivar otros nodos.
- ✓ `Attr`, con este tipo de nodos representamos los atributos de las etiquetas XHTML, es decir, un nodo por cada atributo=valor.
- ✓ `Text`, es el nodo que contiene el texto encerrado por una etiqueta XHTML.
- ✓ `Comment`, representa los comentarios incluidos en la página XHTML.

Los otros tipos de nodos pueden ser: `CdataSection`, `DocumentFragment`, `DocumentType`, `EntityReference`, `Entity`, `Notation` y `ProcessingInstruction`.

1.3.- Acceso a los nodos.



Cuando ya se ha construido automáticamente el árbol de nodos del DOM, ya podemos comenzar a utilizar sus funciones para acceder a cualquier nodo del árbol. El acceder a un nodo del árbol, es lo equivalente a acceder a un trozo de la página de nuestro documento. Así que, una vez que hemos accedido a esa parte del documento, ya podemos modificar valores, crear y añadir nuevos elementos, moverlos de sitio, etc.

Para acceder a un nodo específico (elemento XHTML) lo podemos hacer empleando dos métodos: o bien a través de los nodos padre, o bien usando un método de acceso directo. A través de los nodos padre partiremos del nodo raíz e iremos accediendo a los nodos hijo, y así sucesivamente hasta llegar al elemento que deseemos. Y para el método de acceso directo, que por cierto es el método más utilizado, emplearemos funciones del DOM, que nos permiten ir directamente a un elemento sin tener que atravesar nodo a nodo.

Algo muy importante que tenemos que destacar es, que para que podamos acceder a todos los nodos de un árbol, el árbol tiene que estar completamente construido, es decir, cuando la página XHTML haya sido cargada por completo, en ese momento es cuando podremos acceder a cualquier elemento de dicha página.

Consideremos el siguiente ejemplo y veamos las formas de acceso:

```
<input type="text" id="apellidos" name="apellidos" />
```

getElementsByName()

Esta función obtiene una colección, que contiene todos los elementos de la página XHTML cuyo atributo `name` coincida con el indicado como parámetro.

```
var elementos = document.getElementsByName("apellidos");
```

Una colección no es un array, aunque se le parezca mucho, ya que aunque puedas recorrerla y referenciar a sus elementos como un array, no se pueden usar métodos de array, como push o pop, en la colección.

Si sólo tenemos un elemento con `name="apellidos"` para acceder a él haremos: `var elemento = document.getElementsByName("apellidos")[0];`

Por ejemplo, si tuviéramos 3 elementos con el atributo `name="apellidos"` para acceder al segundo elemento haríamos: `var segundo = document.getElementsByName("apellidos")[1]; // recordarte que los arrays comienzan en la posición 0`.

Lo que nos permiten estas colecciones de elementos, es el poder recorrerlas fácilmente empleando un bucle, por ejemplo:

```
for (var j=1; j<document.getElementsByName("apellidos").length; j++)
{     var elemento = document.getElementsByName("apellidos")[j];  …..   }
```

getElementsByTagName()

Esta función es muy similar a la anterior y también devuelve una colección de elementos cuya etiqueta XHTML coincida con la que se pasa como parámetro. Por ejemplo:

```
var elementos = document.getElementsByTagName("input");
// Este array de elementos contendrá todos los elementos input del documento.

var cuarto = document.getElementsByTagName("input")[3];
```

getElementById()

Esta función es la más utilizada, ya que nos permite acceder directamente al elemento por el ID. Entre paréntesis escribiremos la cadena de texto con el ID. Es muy importante que el ID sea único para cada elemento de una misma página. La función nos devolverá únicamente el nodo buscado. Por ejemplo:

```
var elemento= document.getElementById("apellidos");
```

Si tenemos por ejemplo una tabla con `id="datos"` y queremos acceder a todas las celdas de esa tabla, tendríamos que combinar `getElementById` con `getElementsByTagName`. Por ejemplo:

```
var miTabla= document.getElementById("datos");
var filas= miTabla.getElementsByTagName("td");
```

1.4.- Acceso a los nodos de tipo atributo.

Una vez que ya hemos visto cómo acceder a los nodos (elementos XHTML) en un documento, vamos a ver cómo podemos acceder a los nodos de tipo atributo. Para referenciar un atributo, como por ejemplo el atributo `type="text"` del campo "apellidos", emplearemos la colección `attributes`. Dependiendo del navegador, esta colección se podrá cubrir de diferentes maneras y podrán existir muchos pares en la colección, tantos como atributos tenga el elemento. Para buscar el par correcto emplearemos la propiedad `nodeName`, que nos devolverá el nombre del atributo (en minúsculas cuando trabajamos con XHTML), y para acceder a su valor usaremos `nodeValue`.

En el ejemplo:

```
<input type="text" id="apellidos" name="apellidos" />
```

Para imprimir todos los atributos del elemento "apellidos", podríamos hacer un bucle que recorriera todos esos atributos imprimiendo su valor:

```
document.write("<br/>El elemento <b>apellidos</b> contiene los pares atributo -> valor:
<br/>");

for( var x = 0; x < document.getElementById("apellidos").attributes.length; x++ )
{
   var atributo = document.getElementById("apellidos").attributes[x];
   document.write(atributo.nodeName+ " -> "+atributo.nodeValue+"<br/>");
```

```
}
```

También podemos **modificar los valores de un atributo** de un nodo manualmente, por ejemplo:

```
document.getElementById("apellidos").attributes[0].nodeValue="password";
// En este caso hemos modificado el type del campo apellidos y lo hemos puesto de tipo "password".
```

O también:

```
document.getElementById("apellidos").attributes["type"].nodeValue="password";
// hemos puesto el nombre del atributo como referencia en el array de atributos.
```

O también:

```
document.getElementById("apellidos").type="password";
// hemos puesto el atributo como una propiedad del objeto apellidos y lo hemos modificado.
```

El método **setAttribute()** nos permitirá **crear o modificar atributos** de un elemento. Por ejemplo, para ponerle de nuevo al campo "apellidos" **type='text'** y un **value='Cid Blanco'**, haríamos:

```
document.getElementById("apellidos").setAttribute('type','text');
document.getElementById("apellidos").setAttribute('value','Cid Blanco');
```

Si lo que quieres realmente es chequear el valor del atributo y no modificarlo, se puede utilizar **getAttribute()**:

```
var valor = document.getElementById("apellidos").getAttribute('type');
// o también
var valor= document.getElementById("apellidos").type;
```

Y si lo que quieres es eliminar un atributo, lo podemos hacer con **removeAttribute()**:

```
// <div id="contenedor" align="left" width="200px">
document.getElementById("contenedor").removeAttribute("align");
// Obtendremos como resultado: <div id="contenedor" width="200px">
```

En XHTML los atributos se escribirán siempre en minúsculas.

Verdadero. ○ Falso. ○

Cuando escribimos los atributos en HTML, da igual que se pongan en mayúsculas o minúsculas, pero si estamos trabajando con XHTML deberemos escribirlos siempre en minúsculas (por ejemplo type, value, size, etc.).

1.5.- Acceso a los nodos de tipo texto.

Para ver cómo podemos acceder a la información textual de un nodo, nos basaremos en el siguiente ejemplo:

```
<p title="Texto de un párrafo">Esto es un ejemplo de <b>texto HTML<br />
que puedes tener</b> en tu documento.</p>
```

Para poder referenciar el fragmento `"texto HTML"` del nodo `P`, lo que haremos será utilizar la colección `childNodes`. Con la colección `childNodes` accederemos a los nodos hijo de un elemento, ya sean de tipo elemento o texto.

Aquí puedes ver una imagen del árbol DOM para ese elemento en cuestión:

Y el código de JavaScript para mostrar una alerta, con el contenido `"texto HTML"`, sería:

```
window.alert(document.getElementsByTagName("p")[0].childNodes[1].childNodes[0].nodeValue);
```

`childNodes[1]` : selecciona el segundo hijo de **`<p>`** que sería el elemento **`<b>`** (el primer hijo es un nodo de tipo Texto `"Esto es un..."`).
`childNodes[0]` : selecciona el primer hijo del elemento **`<b>`** que es el nodo de texto `"texto HTML"`

En lugar de **`childNodes[0]`** también podríamos haber utilizado **`firstChild`**, el cual nos devuelve el primer hijo de un nodo.

Por ejemplo:

```
window.alert(document.getElementsByTagName("p")[0].childNodes[1].firstChild.nodeValue);
```

El tamaño máximo de lo que se puede almacenar en un nodo de texto, depende del navegador, por lo que muchas veces, si el texto es muy largo, tendremos que consultar varios nodos para ver todo el contenido.

En el DOM de HTML, para acceder al valor de un nodo de texto, o modificarlo, es muy común ver la propiedad **`innerHTML`**. Esta propiedad es de Microsoft al igual que **`outerHTML`**. Aunque esta propiedad está soportada por casi todos los navegadores, y es muy rápida en su uso para hacer modificaciones, del contenido de un `DIV` por ejemplo, se recomienda usar el DOM si es posible.

Alternativas al uso de innerHTML. http://slayeroffice.com/articles/innerHTML_alternatives

Para modificar el contenido del nodo, modificaremos la propiedad **`nodeValue`** y le asignaremos otro valor. Por ejemplo en el caso anterior si hacemos:

```
document.getElementsByTagName("p")[0].childNodes[1].firstChild.nodeValue = "Texto MODIFICADO";
```

Veremos que en la página web se ha cambiado la cadena **`"texto HTML"`**, por **`"Texto MODIFICADO"`**.

También podríamos por ejemplo, mover trozos de texto a otras partes. El siguiente ejemplo mueve el texto `"en tu documento"` a continuación de `"Esto es un ejemplo de"`:

```
document.getElementsByTagName("p")[0].firstChild.nodeValue +=
document.getElementsByTagName("p")[0].childNodes[2].nodeValue;
document.getElementsByTagName("p")[0].childNodes[2].nodeValue="";
```

El resultado obtenido sería:

```
"Esto es un ejemplo de en tu documento.texto HTML
que puedes tener"
```

1.6.- Creación y borrado de nodos.

La creación y borrado de nodos fue uno de los objetivos para los que se creó el DOM. Podremos crear elementos y luego insertarlos en el DOM, y la actualización quedará reflejada automáticamente por el navegador. También podremos mover nodos ya existentes (como el párrafo del punto 1.4) simplemente insertándolo en cualquier otro lugar del árbol del DOM.

Ten en cuenta que cuando estemos creando nodos de elementos, el elemento debe estar en minúsculas. Aunque en HTML ésto daría igual, el XHTML sí que es sensible a mayúsculas y minúsculas y tendrá que ir, por lo tanto, en minúsculas.

Usaremos los métodos **`createElement()`**, **`createTextNode()`** y **`appendChild()`**, que nos permitirán crear un elemento, crear un nodo de texto y añadir un nuevo nodo hijo.

Ejemplo de creación de un nuevo párrafo, suponiendo que partimos del siguiente código HTML:

```
<p title="Texto de un párrafo" id="parrafito">Esto es un ejemplo de <b>texto HTML<br />
```

```
que puedes tener</b> en tu documento.</p>
```

Para crear el nuevo párrafo haremos:

```
var nuevoParrafo = document.createElement('p');
var nuevoTexto = document.createTextNode('Contenido añadido al párrafo.');
nuevoParrafo.appendChild(nuevoTexto);
document.getElementById('parrafito').appendChild(nuevoParrafo);
```

Y obtendremos como resultado HTML:

```
<p id="parrafito" title="Texto de un párrafo">
Esto es un ejemplo de <b>texto HTML<br>que puedes tener</b>en tu documento.
<p>Contenido añadido al párrafo.</p> </p>
```

Podríamos haber utilizado `insertBefore` en lugar de `appendChild` o, incluso, añadir manualmente el nuevo elemento al final de la colección de nodos `childNodes`. Si usamos `replaceChild`, incluso podríamos sobrescribir nodos ya existentes. También es posible copiar un nodo usando `cloneNode(true)`. Ésto devolverá una copia del nodo, pero no lo añade automáticamente a la colección `childNodes`.

Para eliminar un nodo existente, lo podremos hacer con `element.removeChild(referencia al nodo hijo)`.

Ejemplo de creación de elementos e inserción en el documento:

```
//Creamos tres elementos nuevos: p, b, br
var elementoP = document.createElement('p');
var elementoB = document.createElement('b');
var elementoBR = document.createElement('br');

//Le asignamos un nuevo atributo title al elementoP que hemos creado.
elementoP.setAttribute('title','Parrafo creado desde JavaScript');

//Preparamos los nodos de texto
var texto1 = document.createTextNode('Con JavaScript se ');
var texto2 = document.createTextNode('pueden realizar ');
var texto3 = document.createTextNode('un monton');
var texto4 = document.createTextNode(' de cosas sobre el documento.');

//Añadimos al elemento B los nodos de texto2, elemento BR y texto3.
elementoB.appendChild(texto2);
elementoB.appendChild(elementoBR);
elementoB.appendChild(texto3);

//Añadimos al elemento P los nodos de texto1, elemento B y texto 4.
elementoP.appendChild(texto1);
elementoP.appendChild(elementoB);
elementoP.appendChild(texto4);

//insertamos el nuevo párrafo como un nuevo hijo de nuestro parrafo
document.getElementById('parrafito').appendChild(elementoP);
```

1.7.- Propiedades y métodos de los objetos nodo (DOM nivel 2 W3C).

Propiedades:

Propiedad	Valor	Descripción	IE6Win+	IE5Mac+	Mozilla	Safari
nodeName	String	Varía según el tipo de nodo.	Sí.	Sí.	Sí.	Sí.
nodeValue	String	Varía según el tipo de nodo.	Sí.	Sí.	Sí.	Sí.
nodeType	Integer	Constante que representa cada tipo.	Sí.	Sí.	Sí.	Sí.
parentNode	Object	Referencia al siguiente contenedor más externo.	Sí.	Sí.	Sí.	Sí.
childNodes	Array	Todos los nodos hijos en orden.	Sí.	Sí.	Sí.	Sí.
firstChild	Object	Referencia al primer nodo	Sí.	Sí.	Sí.	Sí.

Propiedad	Valor	Descripción	IE6Win+	IE5Mac+	Mozilla	Safari
		hijo.				
lastChild	Object	Referencia al último nodo hijo.	Sí.	Sí.	Sí.	Sí.
previousSibling	Object	Referencia al hermano anterior según su orden en el código fuente.	Sí.	Sí.	Sí.	Sí.
nextSibling	Object	Referencia al hermano siguiente según su orden en el código fuente.	Sí.	Sí.	Sí.	Sí.
attributes	NodeMap	Array de atributos de los nodos.	Sí.	Algunos.	Sí.	Sí.
ownerDocument	Object	Contiene el objeto document.	Sí.	Sí.	Sí.	Sí.
namespaceURI	String	URI a la definición de namespace.	Sí.	No.	Sí.	Sí.
Prefix	String	Prefijo del namespace.	Sí.	No.	Sí.	Sí.
localName	String	Aplicable a los nodos afectados en el namespace.	Sí.	No.	Sí.	Sí.

Métodos:

Método	Descripción	IE5++	Mozilla	Safari
appendChild(newChild)	Añade un hijo al final del nodo actual.	Sí.	Sí.	Sí.
cloneNode(deep)	Realiza una copia del nodo actual (opcionalmente con todos sus hijos).	Sí.	Sí.	Sí.
hasChildNodes()	Determina si el nodo actual tiene o no hijos (valorboolean).	Sí.	Sí.	Sí.
insertBefore(new, ref)	Inserta un nuevo hijo antes de otro hijo.	Sí.	Sí.	Sí.
removeChild(old)	Borra un hijo.	Sí.	Sí.	Sí.
replaceChild(new, old)	Reemplaza un hijo viejo con el nuevo viejo.	Sí.	Sí.	Sí.
isSupported(feature, version)	Determina cuando el nodo soporta una característica especial.	No.	Sí.	Sí.

Ampliación de información de propiedades y métodos del objeto Nodo.
http://reference.sitepoint.com/javascript/Node

2.- Gestión de eventos.

Caso práctico
Antonio *ya ha visto en anteriores unidades el modelo de registro de Eventos, y los modelos de disparo de eventos. Ha llegado el momento de profundizar un poco más en los eventos, y ver toda la información y posibilidades que nos dan: cómo detener un evento, o su propagación, incompatibilidades entre navegadores en la gestión de eventos, etc.*
Juan *le comenta a* ***Antonio*** *que se centre, sobre todo, en el estudio de los eventos del ratón y del teclado, ya que son los más utilizados, y los que pueden suponer diferencias de gestión según el tipo de navegador web utilizado.*

Como ya te comentábamos en la unidad anterior 5, sin eventos prácticamente no hay scripts. En casi todas las páginas web que incorporan JavaScript, suele haber eventos programados que disparan la ejecución de dichos scripts. La razón es muy simple, JavaScript fue diseñado para añadir interactividad a las páginas: el usuario realiza algo y la página reacciona.

Por lo tanto, JavaScript necesita de alguna forma detectar las acciones del usuario, para saber cuándo reaccionar. También necesita saber las funciones a ejecutar cuando se producen esas acciones.

Cuando el usuario hace algo, se produce un evento. También habrá algunos eventos que no están relacionados directamente con acciones de usuario: por ejemplo, el evento de carga (load) de un documento, que se disparará automáticamente cuando un documento ha sido cargado en el navegador.

También comentábamos que hay diferencias, en lo que es la gestión de eventos, por unos navegadores u otros. Esas diferencias provocan que los programadores de páginas web, tengan que tener mucha precaución con los métodos y propiedades que usan, dependiendo del navegador que ejecutará la página de JavaScript.

Un ejemplo de solución cross-browser (*capacidad que una web, aplicación web, construcción HTML o script del lado del cliente tiene y que permite que sea soportada por todos los navegadores, es decir que se pueda mostrar o ejecutar de forma correcta en cualquier navegador*) para asignar un evento, independientemente del navegador utilizado podría ser:

```
function  crearEvento(elemento, tipoEvento, funcion){
   if (elemento.addEventListener){
      elemento.addEventListener(tipoEvento,  funcion,  false);
   } else if (elemento.attachEvent){
      elemento.attachEvent("on"  +  tipoEvento,  funcion);
   }else{
      elemento["on"  +  tipoEvento]  =  funcion;
   }
}
var miparrafo=document.getElementById("parrafito");
crearEvento(miparrafo, 'click', function(){alert("hola")});
```

La función `crearEvento`, lo primero que intenta hacer es asignar el evento con el método `addEventListener()` del W3C, y que soportan los navegadores más modernos; en el caso de que esa opción falle, intenta asignar el evento usando el método de Internet Explorer; y, si por último esta opción tampoco va, intenta asignar el evento en línea, como un atributo más del objeto.
El método `addEventListener()` no funciona en Internet Explorer 8, pero sí en cambio está ya implementado en Internet Explorer 9. Si tienes Windows XP no podrás usar ese evento ya que la última versión de IE que se puede usar en Windows XP es la 8 y no deja instalar la versión 9. Así que mi recomendación es que utilices las últimas versiones de Mozilla Firefox o Google Chrome.

2.1.- Modelos de eventos.

Vamos a hacer un pequeño repaso de los modelos de eventos, los cuáles fueron detallados en la unidad 5, apartados 3.1 al 3.4 y la fase de disparo de eventos apartado 3.5.

Veíamos que tenemos **4 modelos de registro de eventos:**

- ✓ Modelo de **registro de eventos en línea**:
  - ➔ Los eventos se añaden como un atributo más del objeto.
  - ➔ No es un modelo recomendado hoy en día, porque el código de JavaScript está integrado con el HTML y lo que se intenta conseguir es tener separación entre la estructura y la programación.
  - ➔ Ejemplo: `<A href="pagina.html" onClick="alertar()">Pulsa aqui</a>`
- ✓ Modelo de **registro de eventos tradicional**:
  - ➔ Los eventos se asignan como una propiedad del objeto y fuera de la estructura HTML.
  - ➔ No es un modelo estándar de registro, pero si utilizado ampliamente por Netscape y Microsoft.
  - ➔ Uso de la palabra reservada `this`, para hacer referencia al objeto dónde se programó el evento.
  - ➔ Para asignar un evento se podría hacer: `elemento.evento = hacerAlgo;`
  - ➔ Para eliminar ese evento del objeto: `elemento.evento = null;`
  - ➔ Ejemplo: `document.getElementById("mienlace").onclick = alertar;`
- ✓ Modelo de **registro avanzado de eventos según W3C**:
  - ➔ Es el estándar propuesto por el W3C.
  - ➔ Para asignar un evento se usa `addEventListener()`.
  - ➔ Para eliminar un evento se usa `removeEventListener()`.
  - ➔ Se puede programar cuando queremos que se dispare el evento: en la fase de captura o burbujeo.
  - ➔ Uso de la palabra reservada `this`, para hacer referencia al objeto dónde se programó el evento.
  - ➔ Por ejemplo: `document.getElementById("mienlace").addEventListener('click',alertar,false);`
- ✓ Modelo de **registro de eventos según Microsoft**:
  - ➔ Se parece al utilizado por el W3C.
  - ➔ Para asignar un evento se usa `attachEvent()`.
  - ➔ Para eliminar un evento se usa `detachEvent()`.
  - ➔ Aquí los eventos siempre burbujean, no hay forma de captura.
  - ➔ No se puede usar la palabra reservada `this`, ya que la función es copiada, no referenciada.
  - ➔ El nombre de los eventos comienza por "`on`" + `nombre de evento`.
  - ➔ Por ejemplo: `document.getElementById("mienlace").attachEvent('onclick', alertar);`

Y tenemos **3 modelos propuestos de disparo de eventos**, que clarificarán el orden de disparo de los mismos, cuando se solapen eventos sobre elementos anidados:

- ✓ Modelo de **captura de eventos:**
  - ➔ En este modelo los eventos se van disparando de afuera hacia adentro. Es decir, primero se disparará el evento asignado al elemento exterior, y continúa descendiendo y disparando los eventos que coincidan, hasta llegar al elemento interior.
- ✓ Modelo de **burbujeo de eventos**:
  - ➔ En este modelo los eventos se van disparando desde dentro hacia afuera. Es decir, primero se disparará el evento asignado al elemento interior, y continúa subiendo y disparando los eventos que coincidan, hasta llegar al elemento exterior.
- ✓ Modelo **W3C**:
  - ➔ En este modelo se integran los dos modelos anteriores. Simplemente se realiza la fase de captura de eventos primero y, cuando termina, se realiza la fase de burbujeo. En este modelo cuando registramos un evento con `addEventListener(evento, funcion, true|false)` tenemos la opción de indicar cuándo queremos que se dispare el evento:

- en la **fase de captura (** `, , true` **)**
- en la **fase de burbujeo (** `, , false` **)**

➔ También disponemos de un nuevo método para cancelar eventos con `preventDefault()`, y de un método para detener la propagación de eventos en la fase de burbujeo, con `stopPropagation()`.

2.2.- Tipos de eventos.

Los navegadores anteriores a la versión 4 no tenían acceso al objeto "Evento". Posteriormente, cuando incorporaron los eventos, sólo dejaban asignar algunos tipos de eventos a ciertos elementos HTML, pero, hoy en día, ya podemos aplicar tipos de eventos virtualmente a casi cualquier elemento.

Al principio los eventos se solían asociar en línea en la etiqueta HTML, con un atributo que comenzaba por "on" seguido del tipo del evento, por ejemplo: `onClick=...`, `onSubmit=...`, `onChange=...`, pero hoy en día esa forma de uso está quedando obsoleta, debido a los nuevos modos de registro de eventos propuestos por elW3C, y que soportan ya la mayoría de navegadores modernos.

Eventos comunes en el W3C.

Hay una colección enorme de eventos que pueden ser generados para la mayor parte de elementos HTML:

- ✓ Eventos de ratón.
- ✓ Eventos de teclado.
- ✓ Eventos objetos frame.
- ✓ Eventos de formulario.
- ✓ Eventos de interfaz de usuario.
- ✓ Eventos de mutación (notifican de cualquier cambio en la estructura de un documento).

Algunas categorías y tipos de eventos en el modelo W3C

Categoría	Tipo de Evento	Descripción	Burbujea	Se puede cancelar
Ratón	**click**	Al hacer click sobre un elemento. Un click se define como mousedown y mouseup sobre la misma localización en pantalla.	Sí.	Sí.
	dblclick	Al hacer doble click sobre un elemento.	Sí.	Sí.
	mousedown	Al mantener presionado el botón del ratón sobre un elemento.	Sí.	Sí.
	mousedownmouseup	Al soltar el botón del ratón que estaba sobre un elemento.	Sí.	Sí.
	mouseover	Al pasar el ratón justo sobre un elemento.	Sí.	No.
	mousemove	Cuando el ratón se mueve mientras está sobre un elemento.	Sí.	Sí.
	mouseout	Cuando el ratón sale fuera de un elemento.	Sí.	Sí.
Teclado	**keydown**	Este evento se dispara justo antes del eventokeypress al presionar una tecla.	Sí.	Sí.
	keypress	Este evento se dispara después de keydown al presionar una tecla.	Sí.	Sí.
	keyup	Al soltar una tecla.	Sí.	Sí.

Categoría	Tipo de Evento	Descripción	Burbujea	Se puede cancelar
Frame HTML	**load**	Se dispara cuando se ha terminado de cargar todo el contenido de un documento, incluyendo ventanas, frames, objetos e imágenes.	No.	No.
	unload	Al salir de un documento y modificar el contenido de una ventana.	No.	No.
	abort	Cuando se detiene la carga de un objeto/imagen antes de que esté completamente cargado.	Sí.	No.
	error	Cuando se detiene la carga de un objeto/imagen antes de que esté completamente cargado.	Sí.	No.
	resize	Cuando se redimensiona un documento.	Sí.	No.
	scroll	Cuando nos desplazamos por el documento con scroll.	Sí.	No.

Categorías de eventos formulario, interfaz y mutación y tipos de eventos en el modelo W3C.
http://en.wikipedia.org/wiki/DOM_events

2.3.- El objeto Event.

Generalmente, los manejadores de eventos necesitan información adicional para procesar las tareas que tienen que realizar. Si una función procesa, por ejemplo, el evento `click`, lo más probable es que necesite conocer la posición en la que estaba el ratón en el momento de realizar el click; aunque ésto quizás tampoco sea muy habitual, a no ser que estemos programando alguna utilidad de tipo gráfico.

Lo que sí es más común es tener información adicional en los eventos del teclado. Por ejemplo, cuando pulsamos una tecla nos interesa saber cuál ha sido la tecla pulsada, o si tenemos alguna tecla especial pulsada como Alt, Control, etc.

Para gestionar toda esa información disponemos del objeto `Event`, el cual nos permitirá acceder a esas propiedades adicionales que se generan en los eventos.

Como siempre, los navegadores gestionan de forma diferente los objetos `Event`. Por ejemplo, en las versiones antiguas de Internet Explorer, el objeto `Event` forma parte del objeto `Window`, mientras que en otros navegadores como Firefox, Chrome, etc., para acceder al objeto `Event` lo haremos a través de un parámetro, que escribiremos en la función que gestionará el evento.

Por ejemplo:

```
document.getElementById("unparrafo").addEventListener('click',gestionar,false);

// Este ejemplo también funciona correctamente en la versión 9 de Internet Explorer.

function gestionar(miEvento){
   alert (miEvento.type);  // Mostrará una alerta con el tipo de evento que en este caso es
'click'.
}
```

En el código del ejemplo anterior cuando se produce el evento de click en un párrafo con `id="unparrafo"`, durante la fase de burbujeo, se llamará a la función `gestionar`. En la función `gestionar` hemos creado un argumento que le llamamos `miEvento`, y es justamente **en ese**

argumento que hemos puesto en la función, dónde el navegador de forma automática, pondrá todos los datos referentes al evento que se ha disparado.

Una vez dentro de la función, mostramos una alerta con el tipo de evento (propiedad `type` del objeto `Event`) que se acaba de disparar.

¿En el modelo de registro de eventos de Microsoft podemos configurar que un evento se dispare en la fase de captura?

Verdadero. Falso.

En Internet Explorer no se puede programar que un evento se dispare en la fase de captura, ya que en IE los eventos siempre burbujean.

2.3.1.- Propiedades y métodos del objeto Event.

Veamos una lista de propiedades del objeto **Event**:

Propiedades	Descripción
altKey, ctrlKey, metaKey, shiftKey	Valor booleano que indica si están presionadas alguna de las teclas Alt, Ctrl, Meta o Shift en el momento del evento.
bubbles	Valor booleano que indica si el evento burbujea o no.
button	Valor integer que indica que botón del ratón ha sido presionado o soltado, 0=izquierdo, 2=derecho, 1=medio.
cancelable	Valor booleano que indica si el evento se puede cancelar.
charCode	Indica el carácter Unicode (*estándar de codificación de caracteres diseñado para facilitar el tratamiento informático, transmisión y visualización de textos de múltiples lenguajes y disciplinas técnicas además de textos clásicos de lenguas muertas. El término Unicode proviene de los tres objetivos perseguidos: universalidad, uniformidad y unicidad*) de la tecla presionada.
clientX, clientY	Devuelve las coordenadas de la posición del ratón en el momento del evento.
currentTarget	El elemento al que se asignó el evento. Por ejemplo si tenemos un evento de click en un `divA` que contiene un hijo `divB`. Si hacemos click en `divB`, `currentTarget` referenciará a `divA` (el elemento dónde se asignó el evento) mientras que `target` devolverá `divB`, el elemento dónde ocurrió el evento.
eventPhase	Un valor integer que indica la fase del evento que está siendo procesada. Fase de captura (1), en destino (2) o fase de burbujeo (3).
layerX, layerY	Devuelve las coordenadas del ratón relativas a un elemento posicionado absoluta o relativamente. Si el evento ocurre fuera de un elemento posicionado se usará la esquina superior izquierda del documento.
pageX, pageY	Devuelve las coordenadas del ratón relativas a la esquina superior izquierda de una página.
relatedTarget	En un evento de "`mouseover`" indica el nodo que ha abandonado el ratón. En un evento de "`mouseout`" indica el nodo hacia el que se ha movido el ratón.
screenX, screenY	Devuelve las coordenadas del ratón relativas a la pantalla dónde se disparó el evento.
target	El elemento dónde se originó el evento, que puede diferir del elemento que tenga asignado el evento. Véase `currentTarget`.
timestamp	Devuelve la hora (en milisegundos desde `epoch`) a la que se creó el evento. Por ejemplo cuando se presionó una tecla. No todos los eventos devuelven `timestamp`.
type	Una cadena de texto que indica el tipo de evento "`click`", "`mouseout`", "`mouseover`", etc.
which	Indica el Unicode de la tecla presionada. Idéntico a `charCode`, excepto que esta propiedad también funciona en Netscape 4.

Veamos una lista de métodos del objeto **Event**:

Métodos	Descripción
preventDefault()	Cancela cualquier acción asociada por defecto a un evento.
stopPropagation()	Evita que un evento burbujee. Por ejemplo si tenemos un `divA` que contiene un `divB` hijo. Cuando asignamos un evento de click a `divA`, si hacemos click en `divB`, por defecto se dispararía también el evento en `divA` en la fase de burbujeo. Para evitar esto se puede llamar a `stopPropagation()` en `divB`. Para ello creamos un evento de click en `divB` y le hacemos `stopPropagation()` .

2.3.2.- Eventos del teclado en JavaScript.

Uno de los eventos más complicados de gestionar en JavaScript son los eventos de teclado, debido a que suele haber bastantes incompatibilidades entre navegadores, teclados, idiomas, etc.

Para el teclado disponemos de 3 tipos de eventos: `keydown`, `keypress` y `keyup`. Y además disponemos de dos tipos de teclas: las especiales (Shift, Alt, AltGr, Enter, etc.) y las teclas **normales**, que contienen letras, números, y símbolos.
En el **proceso de pulsación de una tecla** se generan tres eventos seguidos: `keydown`, `keypress` y `keyup`. Y para cada uno de ellos disponemos de las propiedades `keyCode` y `charCode`. Para saber la tecla que se ha pulsado lo más cómodo es acceder al evento `keypress`.

- ✓ `keydown`: se produce al presionar una tecla y mantenerla presionada.
  - ➔ Su comportamiento es el mismo en todos los navegadores.
  - ➔ Propiedad `keyCode`: devuelve el código interno de la tecla.
  - ➔ Propiedad `charCode`: no está definida.
- ✓ `keypress`: se produce en el instante de presionar la tecla.
  - ➔ Propiedad `keyCode`: devuelve el código interno de las teclas especiales, para las teclas normales no está definido.
  - ➔ Propiedad `charCode`: devuelve 0 para las teclas especiales o el código del carácter de la tecla pulsada para las teclas normales.

  (En Internet Explorer `keyCode` devuelve el carácter de la tecla pulsada, y `charCode` no está definido).
- ✓ `keyup`: se produce al soltar una tecla presionada .
  - ➔ Su comportamiento es el mismo en todos los navegadores.
    - Propiedad `keyCode`: devuelve el código interno de la tecla.
    - Propiedad `charCode`: no está definida.

Ejemplo que mueve el foco de un campo de texto a otro, dentro de un formulario, al pulsar la tecla ENTERdentro de cada campo:

```
<form name="formulario" id="formulario">
   <label for="nombre">Nombre: </label><input type="text" id="nombre" name="nombre" /><label
for="apellidos"> Apellidos: </label><input type="text" id="apellidos" name="apellidos"
/><label for="provincia">Provincia: </label><input type="text" id="provincia"
name="provincia" /><input type="button" id="enviar" value="Enviar" />
</form>

<script type="text/javascript">
   function cambiar(evt){
      if (evt.keyCode==13)  // Código de la tecla Enter
         if (this.nextSibling.nextSibling.type=="text")
            this.nextSibling.nextSibling.focus();
   }
   var inputs=document.getElementsByTagName("input");
   for (i=0; i<inputs.length; i++){
      inputs[i].addEventListener("keypress",cambiar,false);
   }
</script>
```

En la estructura HTML del formulario, los campos del formulario no llevan saltos de línea entre unos y otros, por las siguientes razones:

- ✓ `this.nextSibling` - hace referencia al siguiente hermano al actual (la siguiente etiqueta `label` del siguiente campo).
- ✓ `this.nextSibling.nextSibling` - hermano siguiente, al hermano del elemento actual. (será otro elemento `input`. Si pusiéramos un salto de línea entra campos `input` entonces ya ese `this.nextSibling.nextSibling` ya no sería un campo `input` y sería un nodo de texto con el carácter `\n` del salto de línea que hemos puesto como separador de los campos `input`).

Eventos de teclado en formularios

Funcionamiento de la aplicación

Queremos hacer un formulario con tres campos de texto y que al pulsar Intro pase de un campo a otro, excepto cuando el siguiente no es un campo de texto.

Explicación del código fuente

```
<form name="formulario" id="formulario">
   <label for="nombre">Nombre: </label>
   <input type="text" id="nombre" name="nombre" />
   <label for="apellidos">Apellidos: </label>
   <input type="text" id="apellidos" name="apellidos" />
   <label for="provincia">Provincia: </label>
   <input type="text" id="provincia" name="provincia" />
   <input type="button" id="enviar" value="Enviar" />
</form>

<script type="text/javascript">
   function cambiar(evt){
      // evt recibe el código de la tecla pulsada
      if(evt.keyCode==13) //Código de la tecla Enter
      /* el primer nextSibling hace referencia al final de línea del input
         el segundo al elemento label, el tercero a su final de línea y el
         cuarto hace referencia al siguiente input */
         // Si el input siguiente es de texto se sitúa el foco sobre él
         if(this.nextSibling.nextSibling.nextSibling.nextSibling.type=="text")
            this.nextSibling.nextSibling.nextSibling.nextSibling.focus();
   }
   // El array inputs recogerá las etiquetas input del documento
   var inputs=document.getElementByTagName("input");
   // Por cada input existente se crea un listener para cada tecla pulsada, y
   // cuando se produce la pulsación se llama a la función cambiar
   for(i=0;i<inputs.length;i++){
      inputs[i].addEventListener("keypress",cambiar,false);
   }
</script>
```

2.3.3.- Eventos del ratón en JavaScript.

Los eventos del ratón son uno de los eventos más importantes en JavaScript.
Cada vez que un usuario hace click en un elemento, al menos se disparan tres eventos y en el siguiente orden:

1. **`mousedown`**, cuando el usuario presiona el botón del ratón sobre el elemento.
2. **`mouseup`**, cuando el usuario suelta el botón del ratón.
3. **`click`**, cuando el usuario pulsa y suelta el botón sobre el elemento.

En general, los eventos de **`mousedown`** y **`mouseup`** son mucho más útiles que el evento click.

Si por ejemplo presionamos el botón sobre un elemento A, nos desplazamos y soltamos el botón sobre otro elemento B, se detectarán solamente los eventos de **`mousedown`** sobre A y **`mouseup`** sobre B, pero no se detectará el evento de **`click`**. Ésto quizás pueda suponer un problema, dependiendo del tipo de interacción que quieras en tu aplicación. Generalmente a la hora de registrar eventos, se suele hacer para **`mousedown`** y **`mouseup`**, a menos de que quieras el evento de **`click`** y no ningún otro.
El evento de **`dblclick`** no se usa muy a menudo. Incluso si lo usas, tienes que ser muy prudente y no registrar a la vez **`click`** y **`dblclick`** sobre el mismo elemento, para evitar complicaciones.

El evento de **mousemove** funciona bastante bien, aunque tienes que tener en cuenta que la gestión de este evento le puede llevar cierto tiempo al sistema para su procesamiento. Por ejemplo si el ratón se mueve 1 pixel, y tienes programado el evento de **mousemove**, para cada movimiento que hagas, ese evento se disparará, independientemente de si el usuario realiza o no realiza ninguna otra opción. En ordenadores antiguos, ésto puede ralentizar el sistema, ya que para cada movimiento del ratón estaría realizando las tareas adicionales programadas en la función. Por lo tanto se recomienda utilizar este evento sólo cuando haga falta, y desactivarlo cuando hayamos terminado.

Otros eventos adicionales del ratón son los de **mouseover** y **mouseout**, que se producen cuando el ratón entra en la zona del elemento o sale del elemento. Si, por ejemplo, tenemos tres contenedores anidados `divA`, `divB` y `divC`: si programamos un evento de **mouseover** sobre el `divA` y nos vamos moviendo hacia el contenedor interno, veremos que ese evento sigue disparándose cuando estemos sobre `divB` o entremos en `divC`. Ésta reacción se debe al burbujeo de eventos. Ni en `divB` o `divC` tenemos registrado el evento de **mouseover**, pero cuando se produce el burbujeo de dicho evento, se encontrará que tenemos registrado ese evento en el contenedor padre `divA` y por eso se ejecutará.

Muchas veces es necesario saber de dónde procede el ratón y hacia dónde va, y para ello W3C añadió la propiedad **relatedTarget** a los eventos de **mouseover** y **mouseout**. Esta propiedad contiene el elemento desde dónde viene el ratón en el caso de **mouseover**, o el elemento en el que acaba de entrar en el caso de **mouseout**.

Anexo I - Propiedades de destino y origen del objeto Event.

Para saber los botones del ratón que hemos pulsado, disponemos de las propiedades **which** y **button**. Y para detectar correctamente el botón pulsado, lo mejor es hacerlo en los eventos de **mousedown** o **mouseup**. **Which** es una propiedad antigua de Netscape, así que simplemente vamos a citar **button** que es la propiedad propuesta por el W3C:

Los valores de la propiedad **button** pueden ser:

- ✓ **Botón izquierdo**: 0
- ✓ **Botón medio**: 1
- ✓ **Botón derecho**: 2

También es muy interesante conocer la posición en la que se encuentra el ratón, y para ello disponemos de un montón de propiedades que nos facilitan esa información:

- ✓ `clientX`, `clientY`: devuelven las coordenadas del ratón relativas a la ventana.
- ✓ `offsetX`, `offsetY`: devuelven las coordenadas del ratón relativas al objeto destino del evento.
- ✓ `pageX`, `pageY`: devuelven las coordenadas del ratón relativas al documento. Estas coordenadas son las más utilizadas.
- ✓ `screenX`, `screenY`: devuelven las coordenadas del ratón relativas a la pantalla.

Ejemplo que muestra las coordenadas del ratón al moverlo en el documento:

```
<input type="text" id="coordenadas" name="coordenadas" size="12"/>

<script type="text/javascript">

function mostrarCoordenadas(elEvento){
     document.getElementById("coordenadas").value=elEvento.clientX+" : "+elEvento.clientY;
}

document.addEventListener('mousemove',mostrarCoordenadas,false);

</script>
```

Anexo II - Tabla de propiedades de posicionamiento del ratón.

3.- Aplicaciones Cross-Browser (multi-cliente).

Caso práctico
Antonio ha estado programando diferentes eventos en su proyecto y haciendo pruebas en diferentes navegadores, y ha visto que algunas cosas no funcionaban o no lo hacían correctamente. Sobre todo al probar ciertas instrucciones enInternet Explorer, en la versión 8 algunas cosas no funcionan, mientras que sí funcionaban en la versión 9, y claro, si adapta el código para que funcione en Internet Explorer dejará de funcionar en Firefox, Chrome, etc.
Habla con Juan y le pregunta qué puede hacer para que su aplicación pueda funcionar en cualquier tipo de navegador, y Juan le responde que su aplicación tendría que ser cross-browser, es decir, multi-cliente y, de esa forma, el código estaría preparado para ejecutarse en un navegador u otro. Le da una serie de recomendaciones e información para que pueda adaptar las partes de código conflictivas, para que sean totalmente compatibles entre los diferentes navegadores.

Cuando hablamos de aplicaciones cross-browser, nos estamos refiriendo a aplicaciones que se vean exactamente igual en cualquier navegador.

Como bien sabes los navegadores son desarrollados por diferentes empresas de software, cada una con sus propios intereses y, desde siempre, han sido patentes las diferencias entre unos y otros. El W3C define estándares para HTML, CSS y JavaScript, pero muchas veces estas empresas interpretan el estándar de forma distinta, o incluso, a veces, agregan funcionalidades o etiquetas que no están contempladas ni permitidas en el estándar.

El W3C ha ido mejorando y actualizando los estándares, definiendo nuevos niveles del DOM, y por el otro lado, las empresas desarrolladoras de software también se van adaptando, cada vez más, a los estándares propuestos por el W3C.

La historia de cross-browser comenzó con la "guerra de navegadores" al final de 1990 entre Netscape Navigator y Microsoft Internet Explorer y, por lo tanto, también entre JavaScript y JScript (los primeros lenguajes de scripting implementados en estos navegadores respectivamente). Netscape Navigator era el navegador web más usado en ese momento, y Microsoft había sacado Mosaic para crear Internet Explorer 1.0. Nuevas versiones de estos navegadores fueron surgiendo rápidamente, y debido a la feroz competencia entre ellos, muchas veces se añadieron características nuevas, sin ningún tipo de coordinación o control entre fabricantes. La introducción de estas nuevas características a menudo tuvo prioridad sobre la corrección de errores, dando como resultado navegadores inestables, bloqueos, navegadores que no cumplen el estándar y fallos de ejecución, llegando incluso a provocar cierres accidentales de las aplicaciones o del navegador.

Durante todo ese tiempo los programadores de páginas web han sido los encargados de ir parcheando estas diferencias para conseguir que sus aplicaciones se ejecuten de la misma forma en unos u otros navegadores, independientemente de la versión o fabricante utilizado. Estas soluciones que se adaptan a cualquier tipo de navegador son las que se conocen como "soluciones cross-browser".

Las soluciones cross-browser no sólo se aplican a JavaScript, sino que también se pueden aplicar a otras tecnologías como CSS o, incluso, HTML. Lo que se busca por lo tanto es que, esas incompatibilidades o diferencias entre navegadores no sean apreciables por el cliente, y que la página web o aplicación funcione indistintamente en cualquier navegador sin producir fallos o efectos indeseados.

En Internet puedes encontrar múltiples páginas con tablas donde ver las incompatibilidades entre navegadores a nivel de, CSS 2, CSS 3, base del DOM, DOM HTML, eventos del DOM, etc. En estas tablas se muestran todas las características de cada tecnología, se ven las diferentes versiones de

navegadores, y se indica si soportan o no, cada una de las características, propiedades, métodos, etc. A continuación, tienes un enlace muy interesante que es una referencia completa, que te permitirá consultar si ciertas propiedades o métodos que utilizas en JavaScript, son compatibles en todos los navegadores.

Tablas de compatibilidades entre navegadores.

3.1.- Métodos para programar aplicaciones cross-browser (parte I).

A la hora de realizar aplicaciones multi-cliente con JavaScript deberemos tener en cuenta el tipo de navegador que estamos utilizando para que el código se ejecute correctamente. Por ejemplo, si quisiéramos acceder a los nombres de clases CSS empleados por un determinado elemento, dependiendo de si es IE u otro navegador, tendríamos que usar `className` o `classList` respectivamente:

```
if (navigator.appName.indexOf("Explorer") != -1) // Es un navegador IE
{
     // Usaremos className en lugar de classList
     this.parentNode.childNodes[i].className =
this.parentNode.childNodes[i].className.replace(/\bseleccionado\b/,'');
}
else     // Es un navegador W3C
     this.parentNode.childNodes[i].classList.remove("seleccionado");
```

En JavaScript, podemos ejecutar bloques de código dependiendo de una condición determinada. En nuestro caso en el tema de cross-browsing, podríamos comprobar el tipo de navegador que estamos utilizando para ejecutar nuestro código de JavaScript, y dependiendo de eso, ejecutaríamos el código compatible con ese navegador. Por ejemplo, aquí te muestro una función para crear un evento:

```
function crearEvento(elemento, evento, funcion){
     if (typeof elemento.addEventListener !== 'undefined')     {
          // evento compatible con W3C
     }
     else if (typeof elem.attachEvent !== 'undefined')     {
          // evento compatible con Internet Explorer
     }
}
```

La función anterior es una función cross-browser muy simplificada para crear eventos. Pero ¿qué pasaría en el siguiente caso?

```
var elementos = document.getElementsByTagName('div');      // supongamos que nos devuelve 5000
divs
var i, longitud = elementos.length;
for (i = 0; i < longitud; i++)
{
   crearEvento(elementos[i], 'click', function()
     {
          alert('Saludos !');
     });
}
```

En este caso el navegador estaría comprobando 5000 veces `if (typeof elemento.addEventListener !== 'undefined') ….` - una vez para cada elemento de la colección `elementos`, lo cual supone una gran pérdida de tiempo. Estaría mejor si, de alguna manera, pudiéramos decirle al navegador: "Cuando sepas si `addEventListener()` está soportado por este navegador, no continúes comprobando para las otras 4999 iteraciones".

3.1.1.- Métodos para programar aplicaciones cross-browser (parte II).

Para evitar el hacer la comprobación que citamos anteriormente 4999 veces, debemos crear funciones separadas que contengan la lógica de cross-browser, y luego envolviendo esas funciones con otra mayor que devuelva la función apropiada a ejecutar. La parte más ingeniosa de este código es que la parte externa del código de la función (la decisión del tipo de navegador que estamos utilizando) será ejecutada solamente una vez, independientemente del número de llamadas que

hagamos; eso es, en cierto modo, la "*parte condicional de compilación (técnica para mejorar el rendimiento de sistemas de programación, de tal forma que se obtiene código máquina a partir del código fuente con lo que se mejora la ejecución de las aplicaciones. Los compiladores son los encargados de realizar este proceso. El lenguaje JavaScript es un lenguaje interpretado, no compilado)*". Un ejemplo de una función cross-browser para crear eventos podría ser:

```
var crearEvento = function(){
     function w3c_crearEvento(elemento, evento, mifuncion)     {
          elemento.addEventListener(evento, mifuncion, false);
     }

     function ie_crearEvento(elemento, evento, mifuncion)     {
          var fx = function(){
               mifuncion.call(elemento);
          };

          // Cuando usamos attachEvent dejamos de tener acceso
          // al objeto this en mifuncion. Para solucionar eso
          // usaremos el método call() del objeto Function, que nos permitirá
          // asignar el puntero this para su uso dentro de la función. El primer
          // parámetro que pongamos en call será la referencia que se usará como
          // objeto this dentro de nuestra función mifuncion. De esta manera solucionamos el
problema
          // de acceder a this usando attachEvent en Internet Explorer.

          elemento.attachEvent('on' + evento, fx);
     }

     if (typeof window.addEventListener !== 'undefined'){
          return w3c crearEvento;
     }else if (typeof window.attachEvent !== 'undefined')     {
          return ie_crearEvento;
     }
}();     // <= Esta es la parte más crítica - tiene que terminar en ()
```

En este código se ha separado la lógica para IE y los navegadores W3C. Se han desarrollado dos funciones, una para navegadores Internet Explorer, y otra para compatibles W3C. Según el tipo de navegador se devolverá una u otra función. La parte más crítica está en el uso de }(); al final del código. Es importante darse cuenta lo que está pasando aquí: estamos declarando una función `crearEvento`, con el código: `var crearEvento= function() {`, e inmediatamente después, estamos ejecutando esa función al final de su declaración con } (); . De esta forma aunque llamemos a `crearEvento` múltiples veces en nuestro código, sólo se comprobará el tipo de navegador una sola vez con lo que se acelera la ejecución del código. Puedes comprobar el código de la función que se devolvería en tu navegador con: `alert(crearEvento.toString())`.

Para entender mejor la asignación del evento en Internet Explorer, debes recordar que cualquier función en JavaScript es un objeto y como tal tiene sus propiedades y métodos. Entre los métodos de una función podemos tener `toString()` (que nos devuelve el código fuente de una función), métodos `call()`,`apply()`, etc.

Los métodos call y apply en JavaScript.

ANEXO I - W3C DOM Compatibilidades - Eventos

Estas tablas contienen información sobre la compatibilidad de los métodos y propiedades de eventos.

El modulo de eventos W3C DOM no ha sido implementado completamente en Explorer, también existen algunas diferencias en otros módulos. Se añade información sobre la compatibilidad con Netscape 4 porque este sistema es diferente al de W3C y los sistemas Microsoft

Estas páginas contienen las siguientes tablas:

1. **Targets** (currentTarget, from/toElement, originalTarget, relatedTarget, srcElement, target)
2. **Mouse position** (clientX/Y, layerX/Y, offsetX/Y, pageX/Y, screenX/Y, x/y)
3. **Key properties** (altKey/Left, ctrlKey/Left, keyCode, metaKey, modifiers, repeat, shiftKey/Left)
4. **Miscellaneous properties** (button, detail, timeStamp, type, wheelDelta, which)
5. **Event handler registration** (addEventListener, attachEvent, detachEvent, removeEventListener)
6. (Bubbling and canceling; aún no preparado)
7. (Event creation; aún no preparado)

SI	NO	INCORRECTO	CASI

Targets

Propiedad	Eventos	Explorer 5	Explorer 6	Explorer 5.2 MAC	Mozilla 1.75	Safari 1.2	Opera 8	Netscape 4
`currentTarget` The currentTarget property refers to the HTML element currently handling the event. This is the element the event handler has been registered on.	focus keypress mouseover mouseup resize							
		Microsoft no posee una propiedad equivalente						
`fromElement`, `toElement` The from/toElement properties exist only for mouseover and mouseout. They refer to the element the mouse comes from (mouseover) and the element the mouse goes to (mouseout).	mouseover							
		Equivalente a la propiedad `relatedTarget` de W3C						

<table>
<tr><td rowspan="2"><code>relatedTarget</code>
Esta propiedad relativa al evento de un elemento HTML se refiere a desde dónde va el ratón o dónde va</td><td>mouseover</td><td> 
Propiedad equivalente para Microsoft: from / toElement</td></tr>
<tr><td rowspan="4"><code>srcElement</code>
Esta propiedad es el equivalente de target en Microsoft</td><td>focus</td><td>  </td></tr>
<tr><td>keypress</td><td>  </td></tr>
<tr><td>mouseover, mouseup</td><td>  </td></tr>
<tr><td>resize</td><td>  </td></tr>
<tr><td rowspan="4"><code>target</code>
The target property refers to the HTML element the event actually took place on, even when the event handler has been registered not on the HTML element but on one of its ancestors.
Mozilla 1.3 and Safari v73 have obligingly mended their earlier target confusion: text nodes are no longer counted as separate targets.
Unfortunately Safari v85 has reverted to the incorrect behaviour.</td><td>focus</td><td> 
Netscape 4 devuelve el objeto de ventana (window) o el outerHTML completo del nodo (textarea)</td></tr>
<tr><td>keypress</td><td> 
Netscape 4 devuelve nada (document) o el outerHTML completo del nodo (textarea)</td></tr>
<tr><td>mouseover, mouseup</td><td> 
✓ En enlaces, Explorer devuelve el href complete como target/srcElement instanciado de un element A
✓ Netscape 4 no soporta target en mousedown, mouseup and click. Esto es un error muy serio.
✓ Cuando el evento del ratón ocurre en un texto o dentro de los elementos y “burbujea” el contenedor de elementos, Mozilla 1.2 y superiors y Safari v60 y v80 miran el nodo texto como destino del evento. Otros navegadores informan del elemento en línea y no reaccionan a eventos en nodos de texto.</td></tr>
<tr><td>resize</td><td> 
Ignoro la diferencia entre [object Window] y el nodo document. Esto causa muchos problemas en conexiones del documento HTML a la ventana.
Safari devuelve null.</td></tr>
</table>

Mouse position

Selector	IE 5.5	IE6	IE7	IE8	IE9 pr3	FF 3.0	FF 3.5	FF 3.6	FF 4b1	Saf 4 Win	Saf 5 Win	Chrome 4	Chrome 5	Opera 10.10	Opera 10.53	Opera 10.6	Konqueror 4.x
	WindowView properties																
innerWidth and innerHeight The dimensions of the viewport (interior of the browser window)																	
	`window.innerWidth` `window.innerHeight`																
outerWidth and outerHeight The dimensions of the entire browser window (including taskbars and such)																	
	`window.outerWidth` `window.outerHeight`																
pageXOffset and pageYOffset The amount of pixels the entire pages has been scrolled																	
	`window.pageXOffset` `window.pageYOffset`																
screenX and screenY The position of the browser window on the screen																	
	`window.pageXOffset` `window.pageYOffset`																
	Opera calculates the coordinates of the specific tab window relative to the encompassing browser window. This is understandable given its way of working with windows, but strictly speaking it's a bug. It should give the coordinates of the encompassing browser window relative to the screen.																
	ScreenView properties																
availWidth and availHeight The available width and height on the screen (excluding OS taskbars and such)																	
	`screen.availWidth` `screen.availHeight`																
colorDepth The color depth (in bits) of the screen																	
	`sscreen.colorDepth`																
	Firefox 3.6 return 24, while my system is clearly 32; all other browsers agree on that. This bug has occurred before, in a far earlier version of Firefox. Can't remember which one. There was something about Firefox being "really" right because of an alpha channel or something, but as long as all other browsers on the same computer report 32 I continue to see Firefox as the culprit here.																

Selector	IE 5.5	IE6	IE7	IE8	IE9 pr3	FF 3.0	FF 3.5	FF 3.6	FF 4b1	Saf 4 Win	Saf 5 Win	Chrome 4	Chrome 5	Opera 10.10	Opera 10.53	Opera 10.6	Konqueror 4.x
	Besides, these properties are only ever used in web browser statistics programs, and there people have grown used to the “32” even though they don’t understand what it means. Nobody ever uses it in the wild.																
pixelDepth Usually same as colorDepth																	
	`screen.pixelDepth` The difference between colorDepth and pixelDepth is only important on (older?) Unix machines, where old X-clients may allow applications to define their own color scheme. If that’s the case, colorDepth matches the color depth of the application and pixelDepth the color depth of the monitor. In all other cases they’re equal.																
width and height The width and height of the screen																	
	`screen.width` `screen.height`																
	DocumentView and ElementView methods																
elementFromPoint() Returns the element located at the given coordinates																	
	`document.elementFromPoint(100,100)` Which coordinates does elementFromPoint() need? The standard seems to be clientX/Y. Safari 4 and Opera 10.10 need pageX/Y. This method is a godsend for drag and drop scripts. When the user drops the dragged element, figure out what element is located at the drop point and go on from there. No more complicated calculations necessary. However, you need to temporarily hide the dragged object. By definition it's the topmost element on the requested coordinates, and we need to know what's underneath it. The basic trick is: `releaseElement: function(e) { // called onmouseup` `  var evt = e || window.event;` `  draggedObject.style.display = 'none';` `  var receiver = document.elementFromPoint(evt.clientX,evt.clientY);` `  if (receiver.nodeType == 3) { // Opera` `  receiver = receiver.parentNode;` `  }` `  draggedObject.style.display = '';` Now `receiver` contains the element the user dropped the dragged element on.																
getBoundingClientRect() Gets the encompassing rectangle																	
	`x.getBoundingClientRect()` Returns an object that contains the `top`, `left`, `right`, and `bottom` (all relative to the top left of the viewport) of the combined																

Selector	IE 5.5	IE6	IE7	IE8	IE9 pr3	FF 3.0	FF 3.5	FF 3.6	FF 4b1	Saf 4 Win	Saf 5 Win	Chrome 4	Chrome 5	Opera 10.10	Opera 10.53	Opera 10.6	Konqueror 4.x
	rectangle of element `x`. Essentially, the browser calculates all rectangles (see below `getClientRects()`), and `getBoundingClientRect()` returns the lowest (top, left) or highest (bottom, right) values found. IE handles this correctly, provided you accept its incorrect calculation of the individual rectangles. ✓ Firefox doesn't round the top/bottom coordinates.																
getClientRects() Gets the several rectangles of an element																	
	`x.getClientRects()` Returns a list with Rectangle objects that contain the top, left, right, and bottom (all relative to the top left of the viewport) of the rectangles of element x. The trick here is, that an inline element such as an contains one rectangle for every inline box (line), and that all these rectangles are returned. ✓ IE5-7 returns far too many rectangles for the first test paragraphs. The correct number is 5 (for 5 lines), but IE5.5 returns 9 and IE6/7 14. IE8b2 gets this right. ✓ Furthermore, IE5-7 also split up a block-level element such as a <p> into one rectangle per line. This is incorrect: a block-level element should be reported as one rectangle. ✓ Finally, in IE 5-7 the rectangles are off by about two pixels. According to John Resig this is caused by the (invisible, but nonetheless present) borders of the <html> element. ✓ Firefox doesn't round the top/bottom coordinates.																
scrollIntoView() Makes an element scroll into view (Not part of the specification)																	
	`x.scrollIntoView()` Element x scrolls into view. Essentially element x behaves as if it's the target of an #hash: it scrolls to the topmost, leftmost position allowed. ✓ Safari iPhone handles the Y-coordinate correctly, but it also scrolls to the left edge of the page, which can make this method hard to use when the user has zoomed in.																
	ElementView properties																
clientLeft and clientTop The position of the upper left corner of the content field relative to the upper left corner of the entire element (including borders)																	
	`x.clientLeft` `x.clientTop`																

Selector	IE 5.5	IE6	IE7	IE8	IE9 pr3	FF 3.0	FF 3.5	FF 3.6	FF 4b1	Saf 4 Win	Saf 5 Win	Chrome 4	Chrome 5	Opera 10.10	Opera 10.53	Opera 10.6	Konqueror 4.x
clientWidth and clientHeight The width and height of the content field, excluding border and scrollbar, but including padding																	
	`x.clientWidth` `x.clientHeight`																
offsetLeft and offsetTop The left and top position of the element relative to its offsetParent.																	
	`x.offsetLeft` `x.offsetTop`																
	When calculating offsetTop, IE5-7 does not count elements with position: relative as offsetParents, and moves on to the next offsetParent in the chain. offsetLeft is calculated correctly.																
offsetParent The ancestor element relative to which the offsetLeft/Top are calculated.																	
	`x.offsetParent`																
	When calculating the offsetParent of x the browser moves up the DOM tree to x's ancestors until it encounters one of the following elements. That element becomes x's offsetParent. ✓ `<body>` ✓ An element with a position other than static. ✓ A `<table>`, `<th>` or `<td>`, but **only** if `x` has position: static. The `<body>` element does not have an offsetParent. Nonetheless the `<html>` element sometimes enters the offsetParent chain, though never as the offsetParent of the `<body>`. In IE and Opera elements with position: fixed do not have an offsetParent.																
offsetWidth and offsetHeight The width and height of the entire element, including borders																	
	`x.offsetWidth` `x.offsetHeight`																
scrollLeft and scrollTop The amount of pixels the element has scrolled. Read/write.																	
	`x.offsetLeft` `x.scrollTop` `x.scrollTop = 20`																
scrollWidth and scrollHeight The width and height of the																	
	`x.scrollWidth`																

Selector	IE 5.5	IE6	IE7	IE8	IE9 pr3	FF 3.0	FF 3.5	FF 3.6	FF 4b1	Saf 4 Win	Saf 5 Win	Chrome 4	Chrome 5	Opera 10.10	Opera 10.53	Opera 10.6	Konqueror 4.x
entire content field, including those parts that are currently hidden. If there's no hidden content it should be equal to clientX/Y.	`x.scrollHeight` When you scroll the element all the way down, scrollHeight should be equal to scrollTop + clientHeight. If the element has no scrollbars scrollWidth/Height should be equal to clientWidth/Height. ✓ When the element has no scrollbars IE makes the scrollHeight equal to the actual height of the content; and not the height of the element. scrollWidth is correct, except in IE8, where it's 5 pixels off. ✓ Opera gives odd, incorrect values.																

Key event properties

Propiedad	Eventos	Explorer 5	Explorer 6	Explorer 5.2 MAC	Mozilla 1.75	Safari 1.2	Opera 8	Netscape 4
`altKey` Is true when the alt key has been pressed, false when it hasn't.	keydown keyup							
		Opera doesn't react to the Alt key at all						
`altLeft` Is true when the left alt key has been pressed, false when it hasn't.	keydown keyup							
		Opera doesn't react to the Alt key at all						
`charCode`								
`ctrlKey` Is true when the control key has been pressed, false when it hasn't.	keydown keyup							
`ctrlLeft` Is true when the left control key has been pressed, false when it hasn't.	keydown keyup							
		Explorer Windows always returns false						
`keyCode`								
`metalKey` Is true when the meta key has been pressed, false when it hasn't.	keydown keyup				MAC			
		Which key is the meta key? On Mac it's Command, on Windows I don't know (and the browsers don't know, either). Basically this property is only supported by Mozilla on Mac and Safari. ✓ Mozilla Windows and Opera always return false. The other browsers						

Propiedad	Eventos	Explorer 5	Explorer 6	Explorer 5.2 MAC	Mozilla 1.75	Safari 1.2	Opera 8	Netscape 4
		return undefined.						
`modifiers` Returns a bitmask that shows which special keys were pressed.	keydown keyup	modifiers is a bitmask: 1: Alt 2: Ctrl 4: Shift 8: Command Can be combined (ie. 6 = Ctrl + Shift).return undefined.						
`repeat` Is true when the user keeps the key depressed and the system's key repeat starts up.	keydown keyup							
`shiftKey` Is true when the shift key has been pressed, false when it hasn't.	keydown keyup							
`shiftLeft` Is true when the left shift key has been pressed, false when it hasn't.	keydown keyup							

Miscellaneous properties

Propiedad	Eventos	Explorer 5	Explorer 6	Explorer 5.2 MAC	Mozilla 1.75	Safari 1.2	Opera 8	Netscape 4
`button` Because W3C's spec is unworkable, this property has been reduced to **a complete mess**	mousedown				w3c		w3c	
		The Microsoft implementation is the only one that makes sense. button is a bitmask: 1 - Left button 2 - Right button 4 - Middle button They can be combined in Explorer Windows, so clicking on the left and middle button simultaneously gives abutton of 5. Only Microsoft's implementation allows these						

Propiedad	Eventos	Explorer 5	Explorer 6	Explorer 5.2 MAC	Mozilla 1.75	Safari 1.2	Opera 8	Netscape 4
		combinations, the W3C standard doesn't. Fortunately all browsers agree that a right click has a button value of 2. ✓ Mozilla on Mac sees a Shift+Click as a right button click. ✓ All other Mac browsers only detect the left button, even if the mouse has more than one button. ✓ Opera allows you to *disable* right click detection in the JavaScript preferences. **W3C**'s definition is: 0 - Left button 1 - Middle button 2 - Right button 0 should mean "no button pressed", any other meaning is silly. Besides these values cannot be combined into a bitmask: you'll never know whether the left button has been pressed. This definition is very shoddy work. Old Operas used their own values; but Opera 8b has switched to the W3C values. 1 - Left button 2 - Right button 3 - Middle button						
`detail` The detail property gives some more details about the event.	mousedown				incorrect on windows		incorrect	
		'More details' have only been defined for click events and their siblings: here detail gives the total amount of clicks fired in rapid succession. Works perfectly only in Mozilla Mac and Safari. ✓ Mozilla Windows's implementation is closely linked to the function of repeated clicks. The first click inserts the caret, the second one selects the word, the third one the entire line. The fourth click reverts the selection back to a single word, while the fifth click again selects the whole line. detail's values follow this 1-2-3-2-3 sequence exactly and thus show which text has been selected by the user. ✓ Opera always returns 0						
`timeStamp`	mousedown				incomplete			

Propiedad	Eventos	Explorer 5	Explorer 6	Explorer 5.2 MAC	Mozilla 1.75	Safari 1.2	Opera 8	Netscape 4
The timeStamp property returns the Epoch time at which the event took place (at least, I assume so; values on my Windows and Mac computers differ significantly).	mouseup click keydown keyup press	As long as you use timeStamp for comparing it to other timeStamps, and not for obtaining an absolute time, you shouldn't encounter too many problems. ✓ In *Mozilla* it doesn't work for a click event.						
`type` The type property returns the type of the event, without the 'on' prefix.	Any event							
`wheelDelta` How many pixels the mouse wheel scrolls the page.	mousewheel (MS proprietary)							
`which` **Two meanings** 1) For the key events Returns the code of the pressed key. a = 65 etc. 2) For the mouse events Returns the mouse button pressed 1 - Left button 2 - Middle button 3 - Right button	keydown, keyup							incorrect
		It is extremely difficult to detect arrow keys, function keys and Insert/Home/etc. keys. ✓ Netscape 4 gives a key code 97. This is the old Netscape 4 system.						
	mousedown					(prop)button		incomplete
		✓ Netscape 4 cannot detect the middle button. ✓ Safari gives which the same value as button.						

Event handler registration

Propiedad	Eventos	Explorer 5	Explorer 6	Explorer 5.2 MAC	Mozilla 1.75	Safari 1.2	Opera 8	Netscape 4
`addEventListener()` Añade un manejador de eventos a un elemento	click	x.addEventListener('click',doSomething,false) Add an onclick event handler that executes function doSomething() to element x. The true/false flag at the end states whether the event handler should be executed in the capturing or in the bubbling phase.						

Propiedad	Eventos	Explorer 5	Explorer 6	Explorer 5.2 MAC	Mozilla 1.75	Safari 1.2	Opera 8	Netscape 4
`attachEvent()` Añade un manejador de eventos a un elemento	click	x.attachEvent('onclick',doSomething); Add an onclick event handler that executes function doSomething() to element x.						
`detachEvent()` Quita un manejador de eventos de un elemento	click	x.detachEvent('onclick',doSomething); Remove the onclick event handler that executes function doSomething() from element x.						
`removeventListener()` Quita un manejador de eventos de un elemento	click	x.removeEventListener('click',doSomething,false); Remove the onclick event handler that executes function doSomething() from element x.						

Anexo II - HTML DOM Events

HTML DOM events allow JavaScript to register different event handlers on elements in an HTML document.

Events are normally used in combination with functions, and the function will not be executed before the event occurs (such as when a user clicks a button).

Tip: The event model was standardized by the W3C in DOM Level 2.

HTML DOM Events
DOM: Indicates in which DOM Level the property was introduced.

Mouse Events

Property	Description	DOM
onclick	The event occurs when the user clicks on an element	2
ondblclick	The event occurs when the user double-clicks on an element	2
onmousedown	The event occurs when a user presses a mouse button over an element	2
onmousemove	The event occurs when the pointer is moving while it is over an element	2
onmouseover	The event occurs when the pointer is moved onto an element	2
onmouseout	The event occurs when a user moves the mouse pointer out of an element	2
onmouseup	The event occurs when a user releases a mouse button over an element	2

Keyboard Events

Attribute	Description	DOM
onkeydown	The event occurs when the user is pressing a key	2
onkeypress	The event occurs when the user presses a key	2
onkeyup	The event occurs when the user releases a key	2

Frame/Object Events

Attribute	Description	DOM
onabort	The event occurs when an image is stopped from loading before completely loaded (for <object>)	2
onerror	The event occurs when an image does not load properly (for <object>, <body> and <frameset>)	
onload	The event occurs when a document, frameset, or <object> has been loaded	2
onresize	The event occurs when a document view is resized	2
onscroll	The event occurs when a document view is scrolled	2
onunload	The event occurs once a page has unloaded (for <body> and <frameset>)	2

Form Events

Attribute	Description	DOM
onblur	The event occurs when a form element loses focus	2
onchange	The event occurs when the content of a form element, the selection, or the checked state have changed (for <input>, <select>, and <textarea>)	2
onfocus	The event occurs when an element gets focus (for <label>, <input>, <select>, textarea>, and <button>)	2
onreset	The event occurs when a form is reset	2

onselect	The event occurs when a user selects some text (for <input> and <textarea>)	2
onsubmit	The event occurs when a form is submitted	2

Event Object

Constant	Description	DOM
CAPTURING_PHASE	The current event phase is the capture phase (3)	1
AT_TARGET	The current event is in the target phase, i.e. it is being evaluated at the event target (1)	2
BUBBLING_PHASE	The current event phase is the bubbling phase (2)	3
Property	**Description**	**DOM**
bubbles	Returns whether or not an event is a bubbling event	2
cancelable	Returns whether or not an event can have its default action prevented	2
currentTarget	Returns the element whose event listeners triggered the event	2
eventPhase	Returns which phase of the event flow is currently being evaluated	2
target	Returns the element that triggered the event	2
timeStamp	Returns the time (in milliseconds relative to the epoch) at which the event was created	2
type	Returns the name of the event	2
Method	**Description**	**DOM**
initEvent()	Specifies the event type, whether or not the event can bubble, whether or not the event's default action can be prevented	2
preventDefault()	To cancel the event if it is cancelable, meaning that any default action normally taken by the implementation as a result of the event will not occur	2
stopPropagation()	To prevent further propagation of an event during event flow	2

EventTarget Object

Method	Description	DOM
addEventListener()	Allows the registration of event listeners on the event target (IE8 = attachEvent())	2
dispatchEvent()	Allows to send the event to the subscribed event listeners (IE8 = fireEvent())	2
removeEventListener()	Allows the removal of event listeners on the event target (IE8 = detachEvent())	2

EventListener Object

Method	Description	DOM
handleEvent()	Called whenever an event occurs of the event type for which the EventListener interface was registered	2

DocumentEvent Object

Method	Description	DOM
createEvent()		2

MouseEvent/KeyboardEvent Object

Property	Description	DOM
altKey	Returns whether or not the "ALT" key was pressed when an event was triggered	2
button	Returns which mouse button was clicked when an event was triggered	2
clientX	Returns the horizontal coordinate of the mouse pointer, relative to the	2

	current window, when an event was triggered	
clientY	Returns the vertical coordinate of the mouse pointer, relative to the current window, when an event was triggered	2
ctrlKey	Returns whether or not the "CTRL" key was pressed when an event was triggered	2
keyIdentifier	Returns the identifier of a key	3
keyLocation	Returns the location of the key on the advice	3
metaKey	Returns whether or not the "meta" key was pressed when an event was triggered	2
relatedTarget	Returns the element related to the element that triggered the event	2
screenX	Returns the horizontal coordinate of the mouse pointer, relative to the screen, when an event was triggered	2
screenY	Returns the vertical coordinate of the mouse pointer, relative to the screen, when an event was triggered	2
shiftKey	Returns whether or not the "SHIFT" key was pressed when an event was triggered	2

Method	Description	W3C
initMouseEvent()	Initializes the value of a MouseEvent object	2
initKeyboardEvent()	Initializes the value of a KeyboardEvent object	3

ANEXO III - Tabla maestra de compatibilidades

Selector	IE 5.5	IE6	IE7	IE8	IE9 pr3	FF 3.0	FF 3.5	FF 3.6	FF 4b1	Saf 4 Win	Saf 5 Win	Chrome 4	Chrome 5	Opera 10.10	Opera 10.53	Opera 10.6	Konqueror 4.x
DOM Core Manipulación de nodos	The W3C DOM Core module defines how to access, read and manipulate an XML document. Well-formed HTML documents are XML documents, so these methods and properties can be used to completely rewrite any HTML page, if you so wish. Here you find details on how to find elements, how to create new ones, how to read out node information and how to change the structure of the document.																
DOM HTML Manipulación de etiquetas html	Though HTML documents are XML documents, they have a number of special features that the average XML document doesn't have. The W3C DOM HTML module defines these special cases and how to deal with them. Here you find details on getting and setting properties of HTML elements, such as className or id. The innerHTML property is of prime importance to any DOM script.																
DOM CSS Manipulación de hojas de estilo	Alternativa																
	Style sheets are part of the document, too (sort of). The W3C DOM CSS module gives access to style sheets and allows you to change a style sheet. This module contains some browser incompatibilities, but they are of the cute kind. W3C and Microsoft define some different methods and arrays, but some simple object detection allows you to evade these problems.																
CSS Object Model View dimension de los elementos, coordenadas del ratón y miscelánea	Incompleto																
	This specification contains several age-old properties that all browser support but that never have made it to a W3C specification yet.																

ANEXO IV - Métodos apply() y call()

JavaScript define un par de métodos denominados `apply()` y `call()` que son muy útiles para las funciones. Ambos métodos permiten ejecutar una función como si fuera un método de otro objeto. La única diferencia entre los dos métodos es la forma en la que se pasan los argumentos a la función.

El siguiente ejemplo muestra cómo utilizar el método call() para ejecutar una función como si fuera un método del objeto elObjeto:

```
function miFuncion(x) {
  return this.numero + x;
}

var elObjeto = new Object();
elObjeto.numero = 5;

var resultado = miFuncion.call(elObjeto, 4);
alert(resultado);
```

El primer parámetro del método `call()` es el objeto sobre el que se va a ejecutar la función. Como la función se trata como si fuera un método del objeto, la palabra reservada `this` hace referencia al objeto indicado en la llamada a `call()`. De esta forma, si en la función se utiliza `this.numero`, en realidad se está obteniendo el valor de la propiedad `numero` del objeto.

El resto de parámetros del método `call()` son los parámetros que se pasan a la función. En este caso, solamente es necesario un parámetro, que es el número que se sumará a la propiedad `numero` del objeto.

El método `apply()` es idéntico al método `call()`, salvo que en este caso los parámetros se pasan como un array:

```
function miFuncion(x) {
  return this.numero + x;
}
var elObjeto = new Object();
elObjeto.numero = 5;

var resultado = miFuncion.apply(elObjeto, [4]);
alert(resultado);
```

TEMA 7

Contenido

Programación AJAX en javascript.

Caso práctico
*En estos últimos meses, **Antonio** ha realizado un montón de trabajos en el proyecto, y prácticamente ha terminado todas las tareas. **Juan** le dice que todo el trabajo realizado está muy bien, pero que tendría que actualizar algunos de los procesos para darle un toque de modernidad a la aplicación.*
*Entre las mejoras que le recomienda Juan, están la de utilizar efectos, más dinamismo, usar AJAX (JavaScript Asíncrono y XML) en las validaciones de los formularios, o en cierto tipo de consultas, etc. El término AJAX le suena muy complicado a **Antonio**, pero **Juan** lo convence rápidamente para que intente hacerlo, ya que no necesita aprender ningún lenguaje nuevo. Utilizando un nuevo objeto de JavaScript, con sus propiedades y métodos va a poder emplear AJAX en sus aplicaciones actuales.*
*Además, **Juan** lo anima a que se ponga a estudiar rápido el tema de AJAX, ya que al final, le va a dar una pequeña sorpresa, con una librería que le va a facilitar enormemente el programar con AJAX y conseguir dar buenos efectos y mayor dinamismo al proyecto web. Esa librería gratuita tiene el respaldo de grandes compañías a nivel mundial, que la están utilizando actualmente. También cuenta con infinidad de complementos, para que pueda modernizar su web todo lo que quiera, y todo ello programando muy pocas líneas de código y en un tiempo de desarrollo relativamente corto.*

1.- Introducción a AJAX.

Caso práctico
*AJAX es una tecnología crucial en lo que se conoce como web 2.0. A **Antonio** le atrae mucho el tema, ya que ha visto que con AJAX se pueden hacer envíos y consultas al servidor, sin tener que recargar las páginas web o cambiar de página, con lo que se consigue que sea todo más interactivo y adaptado a las nuevas tendencias. **Antonio** analiza la tecnología AJAX, sus orígenes, y el objeto que se utiliza para realizar las peticiones al servidor y gestionar las respuestas. Su directora **Ada**, le facilita unas direcciones muy interesantes con contenidos y ejemplos de algunas aplicaciones AJAX de antiguos proyectos realizados en la empresa.*

El término AJAX (JavaScript Asíncrono y XML) es una técnica de desarrollo web, que permite comunicar el navegador del usuario con el servidor, en un segundo plano. De esta forma, se podrían realizar peticiones al servidor sin tener que recargar la página, y podríamos gestionar esas respuestas, que nos permitirían actualizar los contenidos de nuestra página, sin tener que realizar recargas.

El término AJAX se presentó por primera vez en el artículo "A New Approach to Web Applications", publicado por Jesse James Carrett el 18 de febrero de 2005.

AJAX no es una tecnología nueva. Son realmente muchas tecnologías, cada una destacando por su propio mérito, pero que se unen con los siguientes objetivos:

- ✓ Conseguir una presentación basada en estándares, usando XHTML, CSS y un uso amplio de técnicas del DOM, para poder mostrar la información de forma dinámica e interactiva.
- ✓ Intercambio y manipulación de datos, usando XML y XSLT.
- ✓ Recuperación de datos de forma asíncrona, usando el objeto `XMLHTTPRequest`.
- ✓ Uso de JavaScript, para unir todos los componentes.

Las tecnologías que forman AJAX son:

- ✓ **XHTML** y **CSS**, para la presentación basada en estándares.
- ✓ **DOM**, para la interacción y manipulación dinámica de la presentación.
- ✓ **XML**, **XSLT** y **JSON**, para el intercambio y manipulación de información.
- ✓ **XMLHTTPRequest**, para el intercambio asíncrono de información.
- ✓ **JavaScript**, para unir todos los componentes anteriores.

El modelo clásico de aplicaciones Web funciona de la siguiente forma: la mayoría de las acciones del usuario se producen en la interfaz, disparando solicitudes HTTP al servidor web. El servidor efectúa un proceso (recopila información, realiza las acciones oportunas), y devuelve una página HTML al cliente. Este es un modelo adaptado del uso original de la Web como medio hipertextual, pero a nivel de aplicaciones de software, este tipo de modelo no es necesariamente el más recomendable.

Cada vez que se realiza una petición al servidor, el usuario lo único que puede hacer es esperar, ya que muchas veces la página cambia a otra diferente, y hasta que no reciba todos los datos del servidor, no se mostrará el resultado, con lo que el usuario no podrá interactuar de ninguna manera con el navegador. Con AJAX, lo que se intenta evitar, son esencialmente esas esperas. El cliente podrá hacer solicitudes al servidor, mientras el navegador sigue mostrando la misma página web, y cuando el navegador reciba una respuesta del servidor, la mostrará al cliente y todo ello sin recargar o cambiar de página.

AJAX es utilizado por muchas empresas y productos hoy en día. Por ejemplo, Google utiliza AJAX en aplicaciones como Gmail, Google Suggest, Google Maps.., así como Flickr, Amazon, etc.

Son muchas las razones para usar AJAX:
- ✓ Está basado en estándares abiertos.
- ✓ Su usabilidad.
- ✓ Válido en cualquier plataforma y navegador.
- ✓ Beneficios que aporta a las aplicaciones web.
- ✓ Compatible con Flash.
- ✓ Es la base de la web 2.0.
- ✓ Es independiente del tipo de tecnología de servidor utilizada.
- ✓ Mejora la estética de la web.

1.1.- Requerimientos previos.

A la hora de trabajar con AJAX debemos tener en cuenta una serie de requisitos previos, necesarios para la programación con esta metodología.

Hasta este momento, nuestras aplicaciones de JavaScript no necesitaban de un servidor web para funcionar, salvo en el caso de querer enviar los datos de un formulario y almacenarlos en una base de datos. Es más, todas las aplicaciones de JavaScript que has realizado, las has probado directamente abriéndolas con el navegador o haciendo doble click sobre el fichero .HTML.

Para la programación con AJAX vamos a necesitar de un servidor web, ya que las peticiones AJAX que hagamos, las haremos a un servidor. Los componentes que necesitamos son:
- ✓ Servidor web (apache, ligHTTPd, IIS, etc).
- ✓ Servidor de bases de datos (MySQL, Postgresql, etc).
- ✓ Lenguaje de servidor (PHP, ASP, etc).

Podríamos instalar cada uno de esos componentes por separado, pero muchas veces lo más cómodo es instalar alguna aplicación que los agrupe a todos sin instalarlos de forma individual. Hay varios tipos de aplicaciones de ese tipo, que se pueden categorizar en dos, diferenciadas por el tipo de sistema operativo sobre el que funcionan:
- ✓ servidor LAMP (Linux, Apache, MySQL y PHP).
- ✓ servidor WAMP (Windows, Apache, MySQL y PHP).

Una aplicación de este tipo, muy utilizada, puede ser XAMPP (tanto para Windows, como para Linux).

Esta aplicación podrás instalarla incluso en una memoria USB y ejecutarla en cualquier ordenador, con lo que tendrás siempre disponible un servidor web, para programar tus aplicaciones AJAX.

Servidor XAMPP (Apache, MySQL, PHP). http://www.apachefriends.org/es/xampp.html
Cómo intalar XAMPP http://www.youtube.com/watch?feature=player_embedded&v=LHomeNG8Iz0

Un complemento muy recomendable para la programación con AJAX, es la extensión gratuita **Firebug** de Firefox. Esta extensión es muy útil, ya que con ella podremos detectar errores en las peticiones AJAX, ver los datos que enviamos, en que formato van, qué resultado obtenemos en la petición y un montón de posibilidades más, como inspeccionar todo el DOM de nuestro documento, las hojas de estilo, detectar errores en la programación con JavaScript, etc.

Complemento Firebug para Firefox. http://getfirebug.com/

1.2.- Comunicación asíncrona.

Como ya te comentábamos en la introducción a AJAX, la mayoría de las aplicaciones web funcionan de la siguiente forma:

1. El usuario solicita algo al servidor.
2. El servidor ejecuta los procesos solicitados (búsqueda de información, consulta a una base de datos, lectura de fichero, cálculos numéricos, etc.).
3. Cuando el servidor termina, devuelve los resultados al cliente.

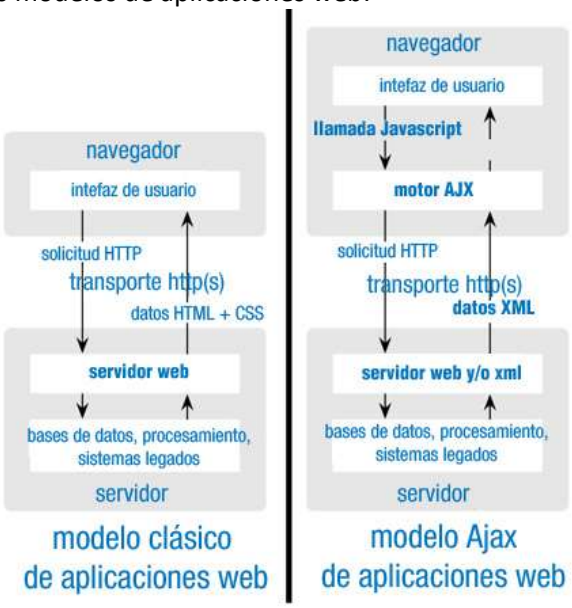
En el paso 2, mientras se ejecutan los procesos en el servidor, el cliente lo único que puede hacer es esperar, ya que el navegador está bloqueado en espera de recibir la información con los resultados del servidor.

Una aplicación AJAX, cambia la metodología de funcionamiento de una aplicación web, en el sentido de que, elimina las esperas y los bloqueos que se producen en el cliente. Es decir, el usuario podrá seguir interactuando con la página web, mientras se realiza la petición al servidor. En el momento de tener una respuesta confirmada del servidor, ésta será mostrada al cliente, o bien se ejecutarán las acciones que el programador de la página web haya definido.

Mira el siguiente gráfico, en el que se comparan los dos modelos de aplicaciones web:

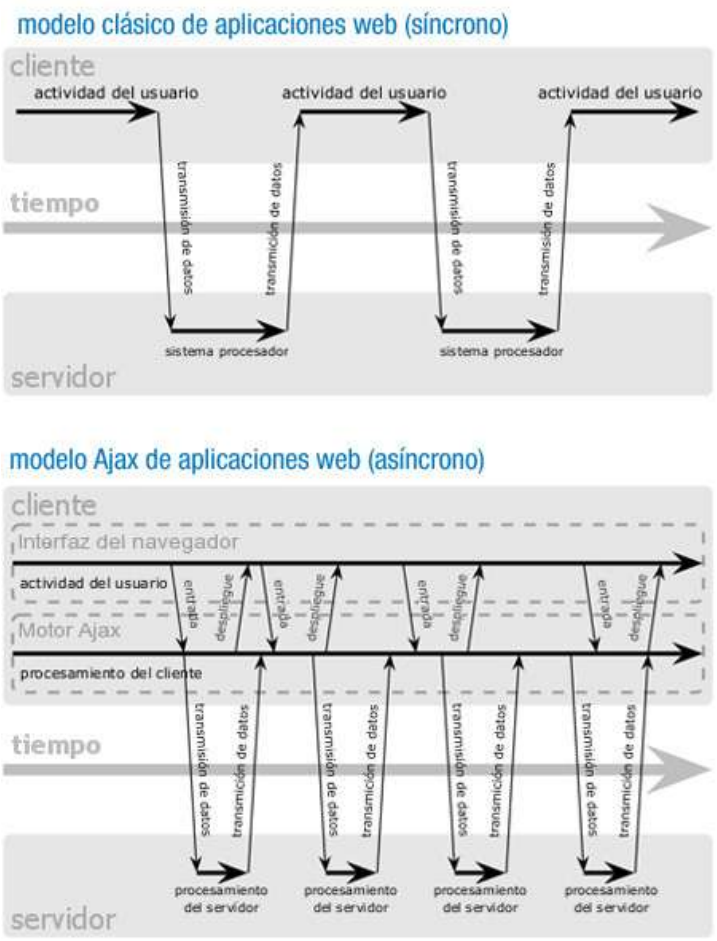
¿Cómo se consigue realizar la petición al servidor sin bloquear el navegador?

Para poder realizar las peticiones al servidor sin que el navegador se quede bloqueado, tendremos que hacer uso del motor AJAX (programado en JavaScript y que generalmente se encuentra en un frame oculto). Este motor se encarga de gestionar las peticiones AJAX del usuario, y de comunicarse con el servidor. Es justamente este motor, el que permite que la interacción suceda de forma asíncrona (independientemente de la comunicación con el servidor). Así, de esta forma, el usuario no tendrá que estar pendiente del icono de indicador de carga del navegador, o viendo una pantalla en blanco.



Cada acción del usuario, que normalmente generaría una petición HTTP al servidor, se va a convertir en una petición AJAX con esa solicitud, y será este motor, el que se encargará de todo el proceso de comunicación y obtención de datos de forma asíncrona con el servidor, y todo ello sin frenar la interacción del usuario con la aplicación.

"Hablar, es el arte de sofocar e interrumpir el pensamiento."
Carlyle, Thomas



¿Según los gráficos anteriores, en qué modelo de aplicación web la actividad del usuario se ve interrumpida o bloqueada por la espera de las respuestas del servidor?

- (•) **Clásico.**
- ( ) AJAX.

En este modelo cuando hacemos una petición al servidor, tendremos que esperar la respuesta del mismo, y mientras esperamos, la página estará bloqueada y no nos dejará hacer ninguna otra cosa.

1.3.- El API XMLHttpRequest.

El corazón de AJAX es una API denominada `XMLHTTPRequest` (XHR), disponible en los lenguajes de scripting en el lado del cliente, tales como JavaScript. Se utiliza para realizar peticiones, HTTP o HTTPS, directamente al servidor web, y para cargar las respuestas directamente en la página del cliente. Los datos que recibamos desde el servidor se podrán recibir en forma de texto plano o texto XML. Estos datos, podrán ser utilizados para modificar el DOM del documento actual, sin tener que recargar la página, o también podrán ser evaluados con JavaScript, si son recibidos en formato JSON.
`XMLHTTPRequest` juega un papel muy importante en la técnica AJAX, ya que sin este objeto, no sería posible realizar las peticiones asíncronas al servidor.

El concepto que está detrás del objeto `XMLHTTPRequest`, surgió gracias a los desarrolladores de Outlook Web Access (de Microsoft), en su desarrollo de Microsoft Exchange Server 2000. La interfaz `IXMLHTTPRequest`, se desarrolló e implementó en la segunda versión de la librería MSXML, empleando este concepto. Con el navegador Internet Explorer 5.0 en Marzo de 1999, se permitió el acceso a dicha interfaz a través de ActiveX.

Posteriormente la fundación Mozilla, desarrolló e implementó una interfaz llamada `nslXMLHTTPRequest`, dentro de su motor Gecko. Esa interfaz, se desarrolló adaptándose lo más posible

a la interfaz implementada por Microsoft. Mozilla creó un envoltorio para usar esa interfaz, a través de un objeto JavaScript, el cuál denominó `XMLHTTPRequest`. El objeto `XMLHTTPRequest` fue accesible en la versión 0.6 de Gecko, en diciembre de 2000, pero no fue completamente funcional, hasta Junio de 2002 con la versión 1.0 de Gecko. El objeto `XMLHTTPRequest`, se convirtió de hecho en un estándar entre múltiples navegadores, como Safari 1.2, Konqueror, Opera 8.0 e iCab 3.0b352 en el año 2005.

El W3C publicó una especificación-borrador para el objeto `XMLHTTPRequest`, el 5 de Abril de 2006. Su objetivo era crear un documento con las especificaciones mínimas de interoperabilidad, basadas en las diferentes implementaciones que había hasta ese momento. La última revisión de este objeto, se realizó en Noviembre de 2009.

Microsoft añadió el objeto `XMLHTTPRequest` a su lenguaje de script, con la versión de Internet Explorer 7.0 en Octubre de 2006.

Con la llegada de las librerías cross-browser (*capacidad que una web, aplicación web, construcciónHTML o script del lado del cliente tiene y que permite que sea soportada por todos los navegadores, es decir que se pueda mostrar o ejecutar de forma correcta en cualquier navegador*) como jQuery, Prototype, etc, los programadores pueden utilizar toda la funcionalidad de `XMLHTTPRequest`, sin codificar directamente sobre la API, con lo que se acelera muchísimo el desarrollo de aplicaciones AJAX.

En febrero de 2008, la W3C publicó otro borrador denominado "`XMLHTTPRequest Nivel 2`". Este nivel consiste en extender la funcionalidad del objeto `XMLHTTPRequest`, incluyendo, pero no limitando, el soporte para peticiones cross-site (*peticiones situadas en diferentes dominios*), gestión de byte streams (*agrupación de bits en unidades que darán lugar a los bytes*), progreso de eventos, etc. Esta última revisión de la especificación, sigue estando en estado "working draft" (borrador), a septiembre de 2010.

Una de las limitaciones de `XMLHTTPRequest` es que, por seguridad, sólo nos deja realizar peticiones AJAX, a las páginas que se encuentren hospedadas en el mismo DOMinio, desde el cual se está realizando la petición AJAX.

"Tan sólo por la educación puede el hombre llegar a ser hombre. El hombre no es más que lo que la educación hace de él."

Kant, Immanuel

1.3.1.- Creación del objeto XMLHttpRequest.

Para poder programar con AJAX, necesitamos crear un objeto del tipo `XMLHTTPRequest`, que va a ser el que nos permitirá realizar las peticiones en segundo plano al servidor web.

Una vez más, nos vamos a encontrar con el problema de Internet Explorer, que, dependiendo de la versión que utilicemos, tendremos que crear el objeto de una manera o de otra. Aquí tienes un ejemplo de una función cross-browser, que devuelve un objeto del tipo XHR (`XMLHTTPRequest`):

```
////////////////////////////////////////////////////////
// Función cross-browser para crear objeto XMLHTTPRequest
////////////////////////////////////////////////////////
function objetoXHR(){
   if (window.XMLHTTPRequest){
      // El navegador implementa la interfaz XHR de forma nativa
      return new XMLHTTPRequest();
   }else if (window.ActiveXObject){
      var versionesIE = new Array('MsXML2.XMLHTTP.5.0', 'MsXML2.XMLHTTP.4.0',
      'MsXML2.XMLHTTP.3.0', 'MsXML2.XMLHTTP', 'Microsoft.XMLHTTP');
      for (var i = 0; i < versionesIE.length; i++){
         try{
            /* Se intenta crear el objeto en Internet Explorer comenzando
            en la versión más moderna del objeto hasta la primera versión.
            En el momento que se consiga crear el objeto, saldrá del bucle
```

```
            devolviendo el nuevo objeto creado. */
            return new ActiveXObject(versionesIE[i]);
         } catch (errorControlado) {}//Capturamos el error,
      }
   }

   /* Si llegamos aquí es porque el navegador no posee ninguna forma de crear el objeto.
      Emitimos un mensaje de error usando el objeto Error.

     Más información sobre gestión de errores en:
     HTTP://www.javascriptkit.com/javatutors/trycatch2.sHTML
   */

   throw new Error("No se pudo crear el objeto XMLHTTPRequest");
}

// para crear un objeto XHR lo podremos hacer con la siguiente llamada.
var objetoAJAX = new objetoXHR();
```

Una opción muy interesante, consiste en hacer una librería llamada, por ejemplo, `funciones.js`, que contenga el código de tus funciones más interesantes como, `crearEvento()`, `objetoXHR()`, etc. De esta manera, irás creando tus propios recursos, con el código de JavaScript que más uses en tus aplicaciones.

"El éxito no se logra sólo con cualidades especiales, es sobre todo un trabajo de constancia, de método y de organización."

Sergent, J.P.

1.3.2.- Métodos del objeto XMLHttpRequest.

El objeto `XMLHTTPRequest` dispone de los siguientes métodos, que nos permitirán realizar peticiones asíncronas al servidor:

Metodo	Descripción
`abort()`	Cancela la solicitud actual.
`getAllResponseHeaders()`	Devuelve la información completa de la cabecera.
`getResponseHeader()`	Devuelve la información específica de la cabecera.
`open(metodo, url, async, usuario, password)`	Especifica el tipo de solicitud, la URL, si la solicitud se debe gestionar de forma asíncrona o no, y otros atributos opcionales de la solicitud. ✓ **método**: indicamos el tipo de solicitud: `GET` o `POST`. ✓ **url**: la dirección del fichero al que le enviamos las peticiones en el servidor. ✓ **async**: `true` (asíncrona) o `false` (síncrona). ✓ **usuario y password**: si fuese necesaria la autenticación en él
`send (datos)`	Envía la solicitud al servidor. ✓ datos: Se usa en el caso de que estemos utilizando el método `POST`, como método de envío. Si usamos `GET`, datos será `null`.
`setRequestHeader()`	Añade el par etiqueta/valor a la cabecera de datos que se enviará al servidor.

Para probar el siguiente código, que incluye una petición AJAX, tienes que hacerlo a través del servidor web. Para ello debes copiar los ficheros del ejemplo, dentro de la raíz del servidor web, en una carpeta a la que llamaremos, por ejemplo, `web/dwec07132`. Arrancaremos el servidor web e iremos a la dirección `HTTP://localhost/web/dwec07132`

Puedes utilizar Firebug, para comprobar cómo se realiza la petición AJAX.

index.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <title>Ejemplo dwec07 - 1.3.2 - AJAX SINCRONO</title>
      <script type="text/javascript" src="funciones.js"></script>
      <script type="text/javascript" src="index.js"></script>
      <style>
         #resultados{
            background: yellow;
         }
      </style>
   </head>
   <body>
      A continuación se cargarán por AJAX los datos recibidos en la solicitud SINCRONA:<br/>
      Esta solicitud tardará 2 segundos aproximadamente, que es el tiempo de ejecución de la
página PHP en el servidor<br/>
      Contenedor resultados:<div id="resultados"></div>
   </body>
</html>
```

index.js

```
////////////////////////////////////////////////////////////////////
// Cuando el documento esté cargado llamamos a la función iniciar().
////////////////////////////////////////////////////////////////////
crearEvento(window,"load",iniciar);
//////////////////////////////////////////////////////

function iniciar(){
   // Creamos un objeto XHR.
   miXHR = new objetoXHR();

   // Cargamos en el objeto con ID resultados el contenido
   // del fichero datos.txt empleando una petición AJAX.
   cargarSync(document.getElementById("resultados"),"fecha.php");
}

////////////////////////////////////////////////////////
// Función cargarSync: carga el contenido de la url
// en el objeto que se le pasa como referencia,
// usando una petición AJAX de forma SINCRONA.
////////////////////////////////////////////////////////
function cargarSync(objeto, url){
   if (miXHR){
      alert("Comenzamos la petición AJAX");

      //Si existe el objeto  miXHR
      miXHR.open('GET', url, false); //Abrimos la url, false=SINCRONA

      // Hacemos la petición al servidor. Como parámetro del método send:
      // null -> cuando usamos GET.
      // cadena con los datos -> cuando usamos POST
      miXHR.send(null);

      //Escribimos la respuesta recibida de la petición AJAX en el objeto DIV
      textoDIV(objeto, miXHR.responseText);

      alert("Terminó la petición AJAX");
   }
}
```

fecha.php

```
<?php
   // retrasamos 2 segundos la ejecución de esta página PHP.
   sleep(2);

   // Mostramos la fecha y hora del servidor web.
   echo "La fecha y hora del Servidor Web: ";
   echo date("j/n/Y G:i:s.");
?>
```

funciones.js

```
////////////////////////////////////////////////////////
// Función cross-browser para crear objeto XMLHttpRequest
////////////////////////////////////////////////////////
function objetoXHR(){
   if (window.XMLHttpRequest){
      // El navegador implementa la interfaz XHR de forma nativa
      return new XMLHttpRequest();
   }else if (window.ActiveXObject){
      var versionesIE = new Array('Msxml2.XMLHTTP.5.0', 'Msxml2.XMLHTTP.4.0',
      'Msxml2.XMLHTTP.3.0', 'Msxml2.XMLHTTP', 'Microsoft.XMLHTTP');

      for (var i = 0; i < versionesIE.length; i++){
         try {
            /*
               Se intenta crear el objeto en Internet Explorer comenzando
               en la versión más moderna del objeto hasta la primera versión.
               En el momento que se consiga crear el objeto, saldrá del bucle
               devolviendo el nuevo objeto creado.
            */
            return new ActiveXObject(versionesIE[i]);
         }catch (errorControlado) {}//Capturamos el error,
      }
   }

   /*
      Si llegamos aquí es porque el navegador no posee ninguna forma de crear el objeto.
      Emitimos un mensaje de error usando el objeto Error.

      Más información sobre gestión de errores en:
      http://www.javascriptkit.com/javatutors/trycatch2.shtml
   */
   throw new Error("No se pudo crear el objeto XMLHttpRequest");
}


////////////////////////////////////////////////////////
// Función cross-browser para añadir Eventos
////////////////////////////////////////////////////////
var crearEvento = function()  {
   function w3c_crearEvento(elemento, evento, mifuncion)    {
      elemento.addEventListener(evento, mifuncion, false);
   }
   function ie_crearEvento(elemento, evento, mifuncion)            {
      var fx = function(){
         mifuncion.call(elemento);
      };

      // Enlazamos el evento con attachEvent. Cuando usamos attachEvent
      // dejamos de tener acceso al objeto this en mifuncion. Para solucionar eso
      // usaremos el método call() del objeto Function, que nos permitirá
      // asignar el puntero this para su uso dentro de la función. El primer
      // parámetro que pongamos en call será la referencia que se usará como
      // objeto this dentro de nuestra función. De esta manera solucionamos el problema
      // de acceder a this usando attachEvent en Internet Explorer.

      elemento.attachEvent('on' + evento, fx);
   }
   if (typeof window.addEventListener !== 'undefined')      {
      return w3c_crearEvento;
   }else if (typeof window.attachEvent !== 'undefined'){
      return ie_crearEvento;
   }
}();  // <= Esta es la parte más crítica - tiene que terminar en ()


////////////////////////////////////////////////////////
// Función cross-browser para modificar el contenido
// de un DIV
////////////////////////////////////////////////////////
function textoDIV(nodo, texto){
   //var nodo = document.getElementById(idObjeto);
   while (nodo.firstChild)
      nodo.removeChild(nodo.firstChild); // Eliminamos todos los hijos de ese objeto.
   // Cuando ya no tenga hijos, agregamos un hijo con el texto que recibe la función.
   nodo.appendChild(document.createTextNode(texto));
}
```

```
function cargarSync(objeto, url) {
   if (miXHR){
      alert("Comenzamos la petición AJAX");

      //Si existe el objeto  miXHR
      miXHR.open('GET', url, false); //Abrimos la url, false=SINCRONA

      // Hacemos la petición al servidor. Como parámetro del método send:
      // null -> cuando usamos GET.
      // cadena con los datos -> cuando usamos POST
      miXHR.send(null);

      //Escribimos la respuesta recibida de la petición AJAX en el objeto DIV
      textoDIV(objeto, miXHR.responseText);

      alert("Terminó la petición AJAX");
   }
}
```

En esta función se realiza una petición AJAX, pero de forma síncrona (comportamiento normal del navegador). En dicha petición se realiza la carga del fichero indicado en la url (debe ser un fichero perteneciente al mismo DOMinio del servidor). La respuesta (**`responseText`**), que obtenemos en esa petición, se coloca en un DIV, con la función personalizada `textoDIV` (su código fuente está en el fichero `funciones.js`).

1.3.3.- Propiedades del objeto XMLHttpRequest.

El objeto **`XMLHTTPRequest`**, dispone de las siguientes propiedades, que nos facilitan información sobre el estado de la petición al servidor, y donde recibiremos los datos de la respuesta devuelta en la petición AJAX:

Propiedad	Descripción
`onreadystatechange`	Almacena una función (o el nombre de una función), que será llamada automáticamente, cada vez que se produzca un cambio en la propiedad `readyState`.
`readyState`	Almacena el estado de la petición `XMLHTTPRequest`. Posibles estados, del 0 al 4: ✓ **0**: solicitud no inicializada. ✓ **1**: conexión establecida con el servidor. ✓ **2**: solicitud recibida. ✓ **3**: procesando solicitud. ✓ **4**: solicitud ya terminada y la respuesta está disponible.
`responseText`	Contiene los datos de respuesta, como una cadena de texto.
`responseXML`	Contiene los datos de respuesta, en formato XML.
`status`	Contiene el estado numérico, devuelto en la petición al servidor (por ejemplo: "`404`" para "No encontrado" o "`200`" para "OK").
`statusText`	Contiene el estado en formato texto, devuelto en la petición al servidor (por ejemplo: "`Not Found`" o "`OK`").

Para probar el siguiente código, que incluye una petición AJAX, tienes que hacerlo a través del servidor web. Para ello debes copiar los siguientes ficheros dentro de la raíz del servidor web, en la carpeta `web/dwes07133`, arrancar el servidor web e ir a la dirección **`HTTP://localhost/web/dwec07133`**

Puedes utilizar Firebug, para comprobar cómo se está realizando la petición AJAX.

index.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
```

```
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <title>Ejemplo dwec07 - 1.3.3 - AJAX ASINCRONO</title>
      <script type="text/javascript" src="funciones.js"></script>
      <script type="text/javascript" src="index.js"></script>
      <style>
         #resultados{
            background: yellow;
         }
      </style>
   </head>
   <body>
      A continuación se cargarán por AJAX los datos recibidos en la solicitud ASINCRONA:<br/>
      Esta solicitud tardará 2 segundos aproximadamente, que es el tiempo de ejecución de la
página PHP en el servidor<br/>
      Contenedor resultados:<div id="resultados"></div>
   </body>
</html>
```

index.js

```
////////////////////////////////////////////////////////////////////
// Cuando el documento esté cargado llamamos a la función iniciar().
////////////////////////////////////////////////////////////////////
crearEvento(window,"load",iniciar);
//////////////////////////////////////////////////////

function iniciar(){
   // Creamos un objeto XHR.
   miXHR = new objetoXHR();

   // Cargamos en el objeto con ID resultados el contenido
   // del fichero datos.txt empleando una petición AJAX.
   cargarAsync(document.getElementById("resultados"),"fecha.php");
}

//////////////////////////////////////////////////////
// Función cargarSync: carga el contenido de la url
// en el objeto que se le pasa como referencia,
// usando una petición AJAX de forma ASINCRONA.
//////////////////////////////////////////////////////
function cargarAsync(objeto, url){
   if (miXHR){
      alert("Comenzamos la petición AJAX");

      //Si existe el objeto  miXHR
      miXHR.open('GET', url, true); //Abrimos la url, true=ASINCRONA

      // Hacemos la petición al servidor. Como parámetro:
      // null -> cuando usamos GET.
      // cadena con los datos -> cuando usamos POST
      miXHR.send(null);

      //Escribimos la respuesta recibida de la petición AJAX en el objeto DIV
      textoDIV(objeto, miXHR.responseText);

      alert("Terminó la petición AJAX");
   }
}
```

fecha.php

```
<?php
   // retrasamos 2 segundos la ejecución de esta página PHP.
   sleep(2);

   // Mostramos la fecha y hora del servidor web.
   echo "La fecha y hora del Servidor Web: ";
   echo date("j/n/Y G:i:s.");
?>
```

funciones.js

```
//////////////////////////////////////////////////////
// Función cross-browser para crear objeto XMLHttpRequest
//////////////////////////////////////////////////////
function objetoXHR(){
   if (window.XMLHttpRequest) {
      // El navegador implementa la interfaz XHR de forma nativa
```

```
      return new XMLHttpRequest();
   }else if (window.ActiveXObject)   {
      var versionesIE = new Array('Msxml2.XMLHTTP.5.0', 'Msxml2.XMLHTTP.4.0',
      'Msxml2.XMLHTTP.3.0', 'Msxml2.XMLHTTP', 'Microsoft.XMLHTTP');

      for (var i = 0; i < versionesIE.length; i++)  {
         try{
            /*
               Se intenta crear el objeto en Internet Explorer comenzando
               en la versión más moderna del objeto hasta la primera versión.
               En el momento que se consiga crear el objeto, saldrá del bucle
               devolviendo el nuevo objeto creado.
            */
            return new ActiveXObject(versionesIE[i]);
         }catch (errorControlado) {}//Capturamos el error,
      }
   }

   /*
      Si llegamos aquí es porque el navegador no posee ninguna forma de crear el objeto.
      Emitimos un mensaje de error usando el objeto Error.

      Más información sobre gestión de errores en:
      http://www.javascriptkit.com/javatutors/trycatch2.shtml
   */

   throw new Error("No se pudo crear el objeto XMLHttpRequest");
}

////////////////////////////////////////////////////////
// Función cross-browser para añadir Eventos
////////////////////////////////////////////////////////
var crearEvento = function(){
   function w3c_crearEvento(elemento, evento, mifuncion)   {
      elemento.addEventListener(evento, mifuncion, false);
   }

   function ie_crearEvento(elemento, evento, mifuncion)    {
      var fx = function(){
         mifuncion.call(elemento);
      };

      // Enlazamos el evento con attachEvent. Cuando usamos attachEvent
      // dejamos de tener acceso al objeto this en mifuncion. Para solucionar eso
      // usaremos el método call() del objeto Function, que nos permitirá
      // asignar el puntero this para su uso dentro de la función. El primer
      // parámetro que pongamos en call será la referencia que se usará como
      // objeto this dentro de nuestra función. De esta manera solucionamos el problema
      // de acceder a this usando attachEvent en Internet Explorer.

      elemento.attachEvent('on' + evento, fx);
   }

   if (typeof window.addEventListener !== 'undefined')     {
      return w3c_crearEvento;
   }else if (typeof window.attachEvent !== 'undefined')    {
      return ie_crearEvento;
   }
}();  // <= Esta es la parte más crítica - tiene que terminar en ()


////////////////////////////////////////////////////////
// Función cross-browser para modificar el contenido
// de un DIV
////////////////////////////////////////////////////////
function textoDIV(nodo, texto){
   //var nodo = document.getElementById(idObjeto);
   while (nodo.firstChild)
      nodo.removeChild(nodo.firstChild); // Eliminamos todos los hijos de ese objeto.
   // Cuando ya no tenga hijos, agregamos un hijo con el texto que recibe la función.
   nodo.appendChild(document.createTextNode(texto));
}
```

```
function cargarAsync(objeto, url) {
   if (miXHR){
```

```
      alert("Comenzamos la petición AJAX");

      //Si existe el objeto  miXHR
      miXHR.open('GET', url, true); //Abrimos la url, true=ASINCRONA

      // Hacemos la petición al servidor. Como parámetro:
      // null -> cuando usamos GET.
      // cadena con los datos -> cuando usamos POST
      miXHR.send(null);

      //Escribimos la respuesta recibida de la petición AJAX en el objeto DIV
      textoDIV(objeto, miXHR.responseText);

      alert("Terminó la petición AJAX");
   }
}
```

En esta función se realiza una petición AJAX, pero de forma asíncrona. En dicha petición se realiza la carga del fichero indicado en la url (debe ser un fichero perteneciente al mismo DOMinio del servidor). La respuesta (**`responseText`**), que obtenemos en esa petición, se coloca en un DIV, con la función personalizada `textoDIV` (su código fuente está en el fichero `funciones.js`). Si ejecutas el ejemplo anterior, verás que no se muestra nada, ya que no hemos gestionado correctamente la respuesta recibida de forma asíncrona. En el siguiente apartado 2, de esta unidad, veremos cómo corregir ese fallo y realizar correctamente esa operación.

2.- Envío y recepción de datos de forma asíncrona.

Caso práctico
*Ahora que **Antonio** ya conoce el objeto `XMLHTTPRequest`, con sus propiedades y métodos, se centra en cómo se realiza la petición al servidor de forma asíncrona, y cómo se gestionan los estados y las respuestas que nos devuelve. También estudia qué formatos tiene para enviar datos al servidor, y en qué formatos va a recibir esos resultados.*
*De entre todos los formatos de recepción de datos, **Juan** recomienda a **Antonio** uno de ellos: el formato JSON. Dicho formato, utiliza la nomenclatura de JavaScript, para enviar los resultados. De esta forma puede utilizar dichos resultados directamente en la aplicación JavaScript, sin tener que realizar prácticamente ningún tipo de conversiones intermedias.*

En el apartado 1.3.3., hemos visto un ejemplo de una petición asíncrona al servidor, pero vimos que éramos incapaces de mostrar correctamente el resultado en el contenedor `resultados`.

```
function cargarAsync(objeto, url) {
   if (miXHR) {
      alert("Comenzamos la petición AJAX");

      //Si existe el objeto  miXHR
      miXHR.open('GET', url, true); //Abrimos la url, true=ASINCRONA

      // Hacemos la petición al servidor. Como parámetro:
      // null -> cuando usamos GET.
      // cadena con los datos -> cuando usamos POST
      miXHR.send(null);

      //Escribimos la respuesta recibida de la petición AJAX en el objeto DIV
      textoDIV(objeto, miXHR.responseText);

      alert("Terminó la petición AJAX");
   }
}
```

Si ejecutas el ejemplo del apartado 1.3.3., verás que no se muestra nada en el contenedor resultados, y la razón es porque, cuando accedemos a la propiedad `miXHR.responseText`, ésta no contiene nada. Eso es debido a la solicitud asíncrona. Si recuerdas, cuando hicimos el ejemplo con una solicitud síncrona, se mostró la alerta de "***Comenzamos la petición AJAX***", aceptaste el mensaje, y justo 2 segundos después recibimos la alerta de "***Terminó la petición AJAX***". En el modo **síncrono**, el navegador cuando hace la petición al servidor, con el método `miXHR.send()`, se queda esperando hasta que termine la solicitud (y por lo tanto nosotros no podemos hacer otra cosa más que esperar ya que el navegador está bloqueado). Cuanto termina la solicitud, pasa a la siguiente línea: `textoDIV(objeto,....)`, y por tanto ya puede mostrar el contenido de `responseText`.

En el modo asíncrono, cuando aceptamos la primera alerta, prácticamente al instante se nos muestra la siguiente alerta. Esto es así, por que el navegador en una petición asíncrona, no espera a que termine esa solicitud, y continúa ejecutando las siguientes instrucciones. Por eso, cuando se hace la llamada con el método `miXHR.send()`, dentro de la función `cargarAsync()`, el navegador sigue ejecutando las dos instrucciones siguientes, sin esperar a que termine la solicitud al servidor. Y es por eso que no se muestra nada, ya que la propiedad `responseText`, no contiene ningún resultado todavía, puesto que la petición al servidor está todavía en ejecución. **¿Cuál será entonces la solución?**

Una primera solución que se nos podría ocurrir, sería la de poner un tiempo de espera o retardo, antes de ejecutar la instrucción `textoDIV`. Esto, lo único que va a hacer, es bloquear todavía más nuestro navegador, y además, tampoco sabemos exactamente lo que va a tardar el servidor web en procesar nuestra solicitud.

La segunda solución, consistiría en detectar cuándo se ha terminado la petición AJAX, y es entonces en ese momento, cuando accederemos a la propiedad `responseText` para coger los resultados. Ésta será la solución, que adoptaremos y que veremos en el apartado 2.1 de esta unidad.

2.1.- Estados de una solicitud asíncrona (parte I).

Cuando se realiza una petición asíncrona, dicha petición va a pasar por diferentes estados (del 0 al 4 - propiedad `readyState` del objeto XHR), independientemente de si el fichero solicitado al servidor, se encuentre o no.

Atención: dependiendo del navegador utilizado, habrá algunos estados que no son devueltos.

Cuando dicha petición termina, tendremos que comprobar cómo lo hizo, y para ello evaluamos la propiedad `status` que contiene el estado devuelto por el servidor: `200`: ***OK***, `404`: ***Not Found***, etc. Si `status` fue **OK** ya podremos comprobar, en la propiedad `responseText` o `responseXML` del objeto XHR, los datos devueltos por la petición.

index.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <title>Ejemplo dwec07 - 2.1 - AJAX ASINCRONO</title>
      <script type="text/javascript" src="funciones.js"></script>
      <script type="text/javascript" src="index.js"></script>
      <style>
         #resultados{
            background: yellow;
         }
      </style>
   </head>
   <body>
      A continuación se cargarán por AJAX los datos recibidos en la solicitud ASINCRONA:<br/>
      Esta solicitud tardará 2 segundos aproximadamente, que es el tiempo de ejecución de la
página PHP en el servidor<br/><br/>
      Contenedor resultados:<div id="resultados"></div>
      Estado de las solicitudes:
      <div id="estado"></div>
   </body>
</html>
```

index.js

```
////////////////////////////////////////////////////////////////////
// Cuando el documento esté cargado llamamos a la función iniciar().
////////////////////////////////////////////////////////////////////
crearEvento(window,"load",iniciar);
////////////////////////////////////////////////////////

function iniciar(){
   // Creamos un objeto XHR.
   miXHR = new objetoXHR();

   // Cargamos el fichero fecha.php de forma asíncrona.
   cargarAsync("fecha.php");
}

////////////////////////////////////////////////////////
// Función cargarAsync: carga el contenido de la url
// usando una petición AJAX de forma ASINCRONA.
////////////////////////////////////////////////////////
function cargarAsync(url){
   if (miXHR){
      alert("Comenzamos la petición AJAX");

      //Si existe el objeto  miXHR
      miXHR.open('GET', url, true); //Abrimos la url, true=ASINCRONA

      // En cada cambio de estado(readyState) se llamará a la función estadoPeticion
      miXHR.onreadystatechange = estadoPeticion;

      // Hacemos la petición al servidor. Como parámetro: null ya que los datos van por GET
```

```
      miXHR.send(null);
   }
}

////////////////////////////////////////////////////////
// Función estadoPeticion: será llamada en cada cambio de estado de la petición.
// Cuando el estado de la petición(readyState) ==4 quiere decir que la petición ha terminado.
// Tendremos que comprobar cómo terminó(status): == 200 encontrado, == 404 no encontrado, etc.
// A partir de ese momento podremos acceder al resultado en responseText o responseXML
////////////////////////////////////////////////////////
function estadoPeticion(){
   switch(this.readyState){
      // Evaluamos el estado de la petición AJAX
      // Vamos mostrando el valor actual de readyState en cada llamada.
      case 0:  document.getElementById('estado').innerHTML += "0 - Sin iniciar.<br/>";
         break;
      case 1: document.getElementById('estado').innerHTML += "1 - Cargando.<br/>";
         break;
      case 2: document.getElementById('estado').innerHTML += "2 - Cargado.<br/>";
         break;
      case 3: document.getElementById('estado').innerHTML += "3 - Interactivo.<br/>";
         break;
      case 4: document.getElementById('estado').innerHTML += "4 - Completado.";
         if (this.status == 200){
            // Si el servidor ha devuelto un OK
            // Escribimos la respuesta recibida de la petición AJAX en el objeto DIV
            textoDIV(document.getElementById("resultados"), this.responseText);
            alert("Terminó la petición AJAX");
         }
         break;
   }
}
```

fecha.php

```
<?php
   // Para que el navegador no haga cache de los datos devueltos por la página PHP.
   header('Cache-Control: no-cache, must-revalidate');
   header('Expires: Mon, 26 Jul 1997 05:00:00 GMT');

   // retrasamos 2 segundos la ejecución de esta página PHP.
   sleep(2);

   // Mostramos la fecha y hora del servidor web.
   echo "La fecha y hora del Servidor Web: ";
   echo date("j/n/Y G:i:s.");
?>
```

funciones.js

```
////////////////////////////////////////////////////////
// Función cross-browser para crear objeto XMLHttpRequest
////////////////////////////////////////////////////////
function objetoXHR(){
   if (window.XMLHttpRequest) {
      // El navegador implementa la interfaz XHR de forma nativa
      return new XMLHttpRequest();
   }else if (window.ActiveXObject)   {
      var versionesIE = new Array('Msxml2.XMLHTTP.5.0', 'Msxml2.XMLHTTP.4.0',
      'Msxml2.XMLHTTP.3.0', 'Msxml2.XMLHTTP', 'Microsoft.XMLHTTP');

      for (var i = 0; i < versionesIE.length; i++)  {
         try{
            /*
               Se intenta crear el objeto en Internet Explorer comenzando
               en la versión más moderna del objeto hasta la primera versión.
               En el momento que se consiga crear el objeto, saldrá del bucle
               devolviendo el nuevo objeto creado.
            */
            return new ActiveXObject(versionesIE[i]);
         }catch (errorControlado) {}//Capturamos el error,
      }
   }

   /*
      Si llegamos aquí es porque el navegador no posee ninguna forma de crear el objeto.
      Emitimos un mensaje de error usando el objeto Error.
```

```
      Más información sobre gestión de errores en:
      http://www.javascriptkit.com/javatutors/trycatch2.shtml
   */

   throw new Error("No se pudo crear el objeto XMLHttpRequest");
}

//////////////////////////////////////////////////////////
// Función cross-browser para añadir Eventos
//////////////////////////////////////////////////////////
var crearEvento = function(){
   function w3c_crearEvento(elemento, evento, mifuncion)   {
      elemento.addEventListener(evento, mifuncion, false);
   }

   function ie_crearEvento(elemento, evento, mifuncion)    {
      var fx = function(){
         mifuncion.call(elemento);
      };

      // Enlazamos el evento con attachEvent. Cuando usamos attachEvent
      // dejamos de tener acceso al objeto this en mifuncion. Para solucionar eso
      // usaremos el método call() del objeto Function, que nos permitirá
      // asignar el puntero this para su uso dentro de la función. El primer
      // parámetro que pongamos en call será la referencia que se usará como
      // objeto this dentro de nuestra función. De esta manera solucionamos el problema
      // de acceder a this usando attachEvent en Internet Explorer.

      elemento.attachEvent('on' + evento, fx);
   }

   if (typeof window.addEventListener !== 'undefined')     {
      return w3c_crearEvento;
   }else if (typeof window.attachEvent !== 'undefined')    {
      return ie_crearEvento;
   }
}();  // <= Esta es la parte más crítica - tiene que terminar en ()


//////////////////////////////////////////////////////////
// Función cross-browser para modificar el contenido
// de un DIV
//////////////////////////////////////////////////////////
function textoDIV(nodo, texto){
   //var nodo = document.getElementById(idObjeto);
   while (nodo.firstChild)
      nodo.removeChild(nodo.firstChild); // Eliminamos todos los hijos de ese objeto.
   // Cuando ya no tenga hijos, agregamos un hijo con el texto que recibe la función.
   nodo.appendChild(document.createTextNode(texto));
}
```

```
//////////////////////////////////////////////////////////
// Función cargarAsync: carga el contenido de la url usando una petición AJAX de forma
// ASINCRONA.
//////////////////////////////////////////////////////////
function cargarAsync(url){
   if (miXHR){
      alert("Comenzamos la petición AJAX");

      //Si existe el objeto  miXHR
      miXHR.open('GET', url, true); //Abrimos la url, true=ASINCRONA

      // En cada cambio de estado(readyState) se llamará a la función estadoPeticion
      miXHR.onreadystatechange = estadoPeticion;

      // Hacemos la petición al servidor. Como parámetro: null ya que los datos van por GET
      miXHR.send(null);
   }
}

//////////////////////////////////////////////////////////
// Función estadoPeticion: será llamada en cada cambio de estado de la petición.
// Cuando el estado de la petición(readyState) ==4 quiere decir que la petición ha terminado.
// Tendremos que comprobar cómo terminó(status): == 200 encontrado, == 404 no encontrado, etc.
// A partir de ese momento podremos acceder al resultado en responseText o responseXML
//////////////////////////////////////////////////////////
```

```
function estadoPeticion(){
   switch(this.readyState){
      // Evaluamos el estado de la petición AJAX
      // Vamos mostrando el valor actual de readyState en cada llamada.
      case 0:  document.getElementById('estado').innerHTML += "0 - Sin iniciar.<br/>";
         break;
      case 1: document.getElementById('estado').innerHTML += "1 - Cargando.<br/>";
         break;
      case 2: document.getElementById('estado').innerHTML += "2 - Cargado.<br/>";
         break;
      case 3: document.getElementById('estado').innerHTML += "3 - Interactivo.<br/>";
         break;
      case 4: document.getElementById('estado').innerHTML += "4 - Completado.";
         if (this.status == 200){
            // Si el servidor ha devuelto un OK
            // Escribimos la respuesta recibida de la petición AJAX en el objeto DIV
            textoDIV(document.getElementById("resultados"), this.responseText);
            alert("Terminó la petición AJAX");
         }
         break;
   }
}
```

2.2.- Estados de una solicitud asíncrona (parte II).

El código del apartado 2.1, fue programado con fines didácticos para mostrar los 4 estados de la petición. El ejemplo completo, con una imagen animada indicadora de la actividad AJAX, se muestra a continuación:

index.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <title>Ejemplo dwec07 - 2.2 - AJAX ASINCRONO</title>
      <script type="text/javascript" src="funciones.js"></script>
      <script type="text/javascript" src="index.js"></script>
      <style>
         #resultados{
            background: yellow;
         }
      </style>
   </head>
   <body>
      A continuación se cargarán por AJAX los datos recibidos en la solicitud ASINCRONA:<br/>
      Esta solicitud tardará 2 segundos aproximadamente, que es el tiempo de ejecución de la
página PHP en el servidor<br/><br/>
      Contenedor resultados:<div id="resultados"></div>
      <div id="indicador"></div>
   </body>
</html>
```

index.js

```
////////////////////////////////////////////////////////////////////
// Cuando el documento esté cargado llamamos a la función iniciar().
////////////////////////////////////////////////////////////////////
crearEvento(window,"load",iniciar);
////////////////////////////////////////////////////////

function iniciar(){
   // Creamos un objeto XHR.
   miXHR = new objetoXHR();

   // Cargamos el fichero fecha.php de forma asíncrona.
   cargarAsync("fecha.php");
}

////////////////////////////////////////////////////////
// Función cargarAsync: carga el contenido de la url
// usando una petición AJAX de forma ASINCRONA.
////////////////////////////////////////////////////////
function cargarAsync(url){
   if (miXHR){
```

```
      // Activamos el indicador Ajax antes de realizar la petición.
      document.getElementById("indicador").innerHTML="<img src='ajax-loader.gif'/>";

      //Si existe el objeto  miXHR
      miXHR.open('GET', url, true); //Abrimos la url, true=ASINCRONA

      // En cada cambio de estado(readyState) se llamará a la función estadoPeticion
      miXHR.onreadystatechange = estadoPeticion;

      // Hacemos la petición al servidor. Como parámetro: null ya que los datos van por GET
      miXHR.send(null);
   }
}

////////////////////////////////////////////////////////
// Función estadoPeticion: será llamada en cada cambio de estado de la petición.
// Cuando el estado de la petición(readyState) ==4 quiere decir que la petición ha terminado.
// Tendremos que comprobar cómo terminó(status): == 200 encontrado, == 404 no encontrado, etc.
// A partir de ese momento podremos acceder al resultado en responseText o responseXML
////////////////////////////////////////////////////////
function estadoPeticion(){
   // Haremos la comprobación en este orden ya que primero tiene que llegar readyState==4
   // y por último se comprueba el status devuelto por el servidor==200.
   if ( this.readyState==4 && this.status == 200 )  {
      // Desactivamos el indicador AJAX.
      document.getElementById("indicador").innerHTML="";

      // Metemos en el contenedor resultados las respuestas de la petición AJAX.
      textoDIV(document.getElementById("resultados"), this.responseText);
   }
}
```

fecha.php

```
<?php
   // Para que el navegador no haga cache de los datos devueltos por la página PHP.
   header('Cache-Control: no-cache, must-revalidate');
   header('Expires: Mon, 26 Jul 1997 05:00:00 GMT');

   // retrasamos 2 segundos la ejecución de esta página PHP.
   sleep(2);

   // Mostramos la fecha y hora del servidor web.
   echo "La fecha y hora del Servidor Web: ";
   echo date("j/n/Y G:i:s.");
?>
```

funciones.js

```
////////////////////////////////////////////////////////
// Función cross-browser para crear objeto XMLHttpRequest
////////////////////////////////////////////////////////
function objetoXHR(){
   if (window.XMLHttpRequest){
      // El navegador implementa la interfaz XHR de forma nativa
      return new XMLHttpRequest();
   }else if (window.ActiveXObject){
      var versionesIE = new Array('Msxml2.XMLHTTP.5.0', 'Msxml2.XMLHTTP.4.0',
      'Msxml2.XMLHTTP.3.0', 'Msxml2.XMLHTTP', 'Microsoft.XMLHTTP');

      for (var i = 0; i < versionesIE.length; i++){
         try{
         /*
            Se intenta crear el objeto en Internet Explorer comenzando
            en la versión más moderna del objeto hasta la primera versión.
            En el momento que se consiga crear el objeto, saldrá del bucle
            devolviendo el nuevo objeto creado.
         */
         return new ActiveXObject(versionesIE[i]);
         }catch (errorControlado) {}//Capturamos el error,
      }
   }

   /*
      Si llegamos aquí es porque el navegador no posee ninguna forma de crear el objeto.
      Emitimos un mensaje de error usando el objeto Error.

      Más información sobre gestión de errores en:
```

```
      http://www.javascriptkit.com/javatutors/trycatch2.shtml
   */

   throw new Error("No se pudo crear el objeto XMLHttpRequest");
}


//////////////////////////////////////////////////////
// Función cross-browser para añadir Eventos
//////////////////////////////////////////////////////
var crearEvento = function(){
   function w3c_crearEvento(elemento, evento, mifuncion){
      elemento.addEventListener(evento, mifuncion, false);
   }

   function ie_crearEvento(elemento, evento, mifuncion){
      var fx = function(){
         mifuncion.call(elemento);
      };

      // Enlazamos el evento con attachEvent. Cuando usamos attachEvent
      // dejamos de tener acceso al objeto this en mifuncion. Para solucionar eso
      // usaremos el método call() del objeto Function, que nos permitirá
      // asignar el puntero this para su uso dentro de la función. El primer
      // parámetro que pongamos en call será la referencia que se usará como
      // objeto this dentro de nuestra función. De esta manera solucionamos el problema
      // de acceder a this usando attachEvent en Internet Explorer.

      elemento.attachEvent('on' + evento, fx);
   }

   if (typeof window.addEventListener !== 'undefined'){
      return w3c_crearEvento;
   }

   else if (typeof window.attachEvent !== 'undefined'){
      return ie_crearEvento;
   }
}();  // <= Esta es la parte más crítica - tiene que terminar en ()


//////////////////////////////////////////////////////
// Función cross-browser para modificar el contenido
// de un DIV
//////////////////////////////////////////////////////
function textoDIV(nodo, texto){
   //var nodo = document.getElementById(idObjeto);
   while (nodo.firstChild)
      nodo.removeChild(nodo.firstChild); // Eliminamos todos los hijos de ese objeto.
   // Cuando ya no tenga hijos, agregamos un hijo con el texto que recibe la función.
   nodo.appendChild(document.createTextNode(texto));
}
```

ajax-loader.gif

```
//////////////////////////////////////////////////////
// Función cargarAsync: carga el contenido de la url
// usando una petición AJAX de forma ASINCRONA.
//////////////////////////////////////////////////////
function cargarAsync(url){
   if (miXHR){
      // Activamos el indicador AJAX.
      document.getElementById("indicador").innerHTML="<img src='AJAX-loader.gif'/>";

      //Si existe el objeto  miXHR
      miXHR.open('GET', url, true); //Abrimos la url, true=ASINCRONA

      // En cada cambio de estado(readyState) se llamará a la función estadoPeticion
      miXHR.onreadystatechange = estadoPeticion;

      // Hacemos la petición al servidor. Como parámetro: null ya que los datos van por GET
      miXHR.send(null);
   }
```

```
}

////////////////////////////////////////////////////////
// Función estadoPeticion: será llamada en cada cambio de estado de la petición.
// Cuando el estado de la petición(readyState) ==4 quiere decir que la petición ha terminado.
// Tendremos que comprobar cómo terminó(status): == 200 encontrado, == 404 no encontrado, etc.
// A partir de ese momento podremos acceder al resultado en responseText o responseXML
////////////////////////////////////////////////////////
function estadoPeticion(){
   // Haremos la comprobación en este orden ya que primero tiene que llegar readyState==4
   // y por último se comprueba el status devuelto por el servidor==200.
   if ( this.readyState==4 && this.status == 200 ){
      // Desactivamos el indicador AJAX.
      document.getElementById("indicador").innerHTML="";

      // Metemos en el contenedor resultados las respuestas de la petición AJAX.
      textoDIV(document.getElementById("resultados"), this.responseText);
   }
}
```

2.3.- Envío de datos usando método GET.

Vamos a ver un ejemplo, en el que se realiza una petición AJAX a la página `procesar.php`, pasando dos parámetros: nombre y apellidos, usando el método **`GET`**.

index.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <title>Ejemplo dwec07 - 2.3 - AJAX ASINCRONO GET - POST</title>
      <script type="text/javascript" src="funciones.js"></script>
      <script type="text/javascript" src="index.js"></script>
      <style>
         #resultados{
            background: yellow;
         }
      </style>
   </head>
   <body>
      A continuación se cargarán por AJAX los datos recibidos en la solicitud ASINCRONA:<br/>
      Contenedor resultados:<div id="resultados"></div>
      <div id="indicador"></div>
   </body>
</html>
```

index.js

```
////////////////////////////////////////////////////////////////////
// Cuando el documento esté cargado llamamos a la función iniciar().
////////////////////////////////////////////////////////////////////
crearEvento(window,"load",iniciar);
////////////////////////////////////////////////////////

function iniciar(){
   // Creamos un objeto XHR.
   miXHR = new objetoXHR();

   // Cargamos de forma asíncrona el texto que nos devuelve la página
   // procesar.php con los parámetros indicados en la URL
   cargarAsync("procesar.php?nombre=Teresa&apellidos=Blanco Ferreiro");
}

////////////////////////////////////////////////////////
// Función cargarAsync: carga el contenido de la url
// usando una petición AJAX de forma ASINCRONA.
////////////////////////////////////////////////////////
function cargarAsync(url){
   if (miXHR){
      // Activamos el indicador Ajax antes de realizar la petición.
      document.getElementById("indicador").innerHTML="<img src='ajax-loader.gif'/>";

      //Si existe el objeto  miXHR
      miXHR.open('GET', url, true); //Abrimos la url, true=ASINCRONA

      // En cada cambio de estado(readyState) se llamará a la función estadoPeticion
```

```
      miXHR.onreadystatechange = estadoPeticion;

      // Hacemos la petición al servidor. Como parámetro: null ya que los datos van por GET
      miXHR.send(null);
   }
}

////////////////////////////////////////////////////
// Función estadoPeticion: será llamada en cada cambio de estado de la petición.
// Cuando el estado de la petición(readyState) ==4 quiere decir que la petición ha terminado.
// Tendremos que comprobar cómo terminó(status): == 200 encontrado, == 404 no encontrado, etc.
// A partir de ese momento podremos acceder al resultado en responseText o responseXML
////////////////////////////////////////////////////
function estadoPeticion(){
   // Haremos la comprobación en este orden ya que primero tiene que llegar readyState==4
   // y por último se comprueba el status devuelto por el servidor==200.
   if ( this.readyState==4 && this.status == 200 ){
      // Desactivamos el indicador AJAX.
      document.getElementById("indicador").innerHTML="";

      // Metemos en el contenedor resultados las respuestas de la petición AJAX.
      textoDIV(document.getElementById("resultados"), this.responseText);
   }
}
```

procesar.php

```
<?php
   // Para que el navegador no haga cache de los datos devueltos por la página PHP.
   header('Cache-Control: no-cache, must-revalidate');
   header('Expires: Mon, 26 Jul 1997 05:00:00 GMT');

   // Imprimimos un mensaje con los textos recibidos
   if (isset($_GET['nombre']))
      echo "Saludos desde el servidor: hola {$_GET['nombre']} {$_GET['apellidos']}. ";
   else if (isset($_POST['nombre']))
      echo "Saludos desde el servidor: hola {$_POST['nombre']} {$_POST['apellidos']}. ";

   // Mostramos la fecha y hora del servidor web.
   echo "La fecha y hora del Servidor Web: ";
   echo date("j/n/Y G:i:s");
?>
```

funciones.js

```
////////////////////////////////////////////////////
// Función cross-browser para crear objeto XMLHttpRequest
////////////////////////////////////////////////////
function objetoXHR(){
   if (window.XMLHttpRequest){
      // El navegador implementa la interfaz XHR de forma nativa
      return new XMLHttpRequest();
   }else if (window.ActiveXObject){
      var versionesIE = new Array('Msxml2.XMLHTTP.5.0', 'Msxml2.XMLHTTP.4.0',
      'Msxml2.XMLHTTP.3.0', 'Msxml2.XMLHTTP', 'Microsoft.XMLHTTP');

      for (var i = 0; i < versionesIE.length; i++){
         try{
            /*
               Se intenta crear el objeto en Internet Explorer comenzando
               en la versión más moderna del objeto hasta la primera versión.
               En el momento que se consiga crear el objeto, saldrá del bucle
               devolviendo el nuevo objeto creado.
            */
            return new ActiveXObject(versionesIE[i]);
         }catch (errorControlado) {}//Capturamos el error,
      }
   }

   /* Si llegamos aquí es porque el navegador no posee ninguna forma de crear el objeto.
   Emitimos un mensaje de error usando el objeto Error.

   Más información sobre gestión de errores en:
   http://www.javascriptkit.com/javatutors/trycatch2.shtml */

   throw new Error("No se pudo crear el objeto XMLHttpRequest");
}
```

```
//////////////////////////////////////////////////////
// Función cross-browser para añadir Eventos
//////////////////////////////////////////////////////
var crearEvento = function(){
   function w3c_crearEvento(elemento, evento, mifuncion){
      elemento.addEventListener(evento, mifuncion, false);
   }

   function ie_crearEvento(elemento, evento, mifuncion){
      var fx = function(){
         mifuncion.call(elemento);
      };

      // Enlazamos el evento con attachEvent. Cuando usamos attachEvent
      // dejamos de tener acceso al objeto this en mifuncion. Para solucionar eso
      // usaremos el método call() del objeto Function, que nos permitirá
      // asignar el puntero this para su uso dentro de la función. El primer
      // parámetro que pongamos en call será la referencia que se usará como
      // objeto this dentro de nuestra función. De esta manera solucionamos el problema
      // de acceder a this usando attachEvent en Internet Explorer.

      elemento.attachEvent('on' + evento, fx);
   }

   if (typeof window.addEventListener !== 'undefined'){
      return w3c_crearEvento;
   }else if (typeof window.attachEvent !== 'undefined'){
      return ie_crearEvento;
   }
}();  // <= Esta es la parte más crítica - tiene que terminar en ()


//////////////////////////////////////////////////////
// Función cross-browser para modificar el contenido
// de un DIV
//////////////////////////////////////////////////////
function textoDIV(nodo, texto){
   //var nodo = document.getElementById(idObjeto);
   while (nodo.firstChild)
      nodo.removeChild(nodo.firstChild); // Eliminamos todos los hijos de ese objeto.
   // Cuando ya no tenga hijos, agregamos un hijo con el texto que recibe la función.
   nodo.appendChild(document.createTextNode(texto));
}
```

```
function iniciar(){
   // Creamos un objeto XHR.
   miXHR = new objetoXHR();

   // Cargamos de forma asíncrona el texto que nos devuelve la página
   // procesar.php con los parámetros indicados en la URL
   cargarAsync("procesar.php?nombre=Teresa&apellidos=Blanco Ferreiro");
}

//////////////////////////////////////////////////////
// Función cargarAsync: carga el contenido de la url
// usando una petición AJAX de forma ASINCRONA.
//////////////////////////////////////////////////////
function cargarAsync(url){
   if (miXHR){
      // Activamos el indicador AJAX antes de realizar la petición.
      document.getElementById("indicador").innerHTML="<img src='AJAX-loader.gif'/>";

      //Si existe el objeto  miXHR
      miXHR.open('GET', url, true); //Abrimos la url, true=ASINCRONA

      // En cada cambio de estado(readyState) se llamará a la función estadoPeticion
      miXHR.onreadystatechange = estadoPeticion;

      // Hacemos la petición al servidor. Como parámetro: null ya que los datos van por GET
      miXHR.send(null);
   }
}
//////////////////////////////////////////////////////
// Función estadoPeticion: será llamada en cada cambio de estado de la petición.
// Cuando el estado de la petición(readyState) ==4 quiere decir que la petición ha terminado.
```

```
// Tendremos que comprobar cómo terminó(status): == 200 encontrado, == 404 no encontrado, etc.
// A partir de ese momento podremos acceder al resultado en responseText o responseXML
////////////////////////////////////////////////////////
function estadoPeticion(){
   // Haremos la comprobación en este orden ya que primero tiene que llegar readyState==4
   // y por último se comprueba el status devuelto por el servidor==200.
   if ( this.readyState==4 && this.status == 200 ){
      // Desactivamos el indicador AJAX.
      document.getElementById("indicador").innerHTML="";

      // Metemos en el contenedor resultados las respuestas de la petición AJAX.
      textoDIV(document.getElementById("resultados"), this.responseText);
   }
}
```

En la petición `GET`, los parámetros que pasemos en la solicitud, se enviarán formando parte de la URL. Por ejemplo: `procesar.php?nombre=Teresa&apellidos=Blanco Ferreiro`. Cuando realizamos la petición por el método `GET`, te recordamos una vez más, que pondremos `null` como parámetro del método `send`, ya que los datos son enviados a la página procesar.php, formando parte de la URL: `send(null)`.

2.4.- Envío de datos usando método POST.

Vamos a ver un ejemplo en el que se realiza una petición AJAX, a la página procesar.php pasando dos párametros: nombre y apellidos, usando el método `POST`.

index.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <title>Ejemplo dwec07 - 2.3 - AJAX ASINCRONO GET - POST</title>
      <script type="text/javascript" src="funciones.js"></script>
      <script type="text/javascript" src="index.js"></script>
      <style>
         #resultados{
            background: yellow;
         }
      </style>
   </head>
   <body>
      A continuación se cargarán por AJAX los datos recibidos en la solicitud ASINCRONA:<br/>
      Contenedor resultados:<div id="resultados"></div>
      <div id="indicador"></div>
   </body>
</html>
```

index.js

```
////////////////////////////////////////////////////////////////////
// Cuando el documento esté cargado llamamos a la función iniciar().
////////////////////////////////////////////////////////////////////
crearEvento(window,"load",iniciar);
////////////////////////////////////////////////////////

function iniciar(){
   // Creamos un objeto XHR.
   miXHR = new objetoXHR();

   // Cargamos de forma asíncrona el texto que nos devuelve la página
   // procesar.php
   // En este caso sólo ponemos los parámetros que pasaremos a la página procesar.php
   cargarAsync("nombre=Teresa&apellidos=Blanco Ferreiro");
}

////////////////////////////////////////////////////////
// Función cargarASync: carga el contenido con los parametros
// que se le vana a pasar a la petición AJAX de forma ASINCRONA.
////////////////////////////////////////////////////////
function cargarAsync(parametros){
   if (miXHR){
      // Activamos el indicador Ajax antes de realizar la petición.
```

```
      document.getElementById("indicador").innerHTML="<img src='ajax-loader.gif'/>";

      // Abrimos la conexión al servidor usando método POST y a la página procesar.php
      miXHR.open('POST', "procesar.php", true);   // Abrimos la url, true=ASINCRONA

      // En cada cambio de estado(readyState) se llamará a la función estadoPeticion
      miXHR.onreadystatechange = estadoPeticion;

      // En las peticiones POST tenemos que enviar en la cabecera el Content-Type
      //ya que los datos se envían formando parte de la cabecera.
      miXHR.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

      // Hacemos la petición al servidor con los parámetros: nombre=Teresa&apellidos=Blanco...
      miXHR.send(parametros);
   }
}

////////////////////////////////////////////////////////
// Función estadoPeticion: será llamada en cada cambio de estado de la petición.
// Cuando el estado de la petición(readyState) ==4 quiere decir que la petición ha terminado.
// Tendremos que comprobar cómo terminó(status): == 200 encontrado, == 404 no encontrado, etc.
// A partir de ese momento podremos acceder al resultado en responseText o responseXML
////////////////////////////////////////////////////////
function estadoPeticion(){
   // Haremos la comprobación en este orden ya que primero tiene que llegar readyState==4
   // y por último se comprueba el status devuelto por el servidor==200.
   if ( this.readyState==4 && this.status == 200 ){
      // Desactivamos el indicador AJAX.
      document.getElementById("indicador").innerHTML="";

      // Metemos en el contenedor resultados las respuestas de la petición AJAX.
      textoDIV(document.getElementById("resultados"), this.responseText);
   }
}
```

procesar.php

```
<?php
   // Para que el navegador no haga cache de los datos devueltos por la página PHP.
   header('Cache-Control: no-cache, must-revalidate');
   header('Expires: Mon, 26 Jul 1997 05:00:00 GMT');

   // Imprimimos un mensaje con los textos recibidos
   if (isset($_GET['nombre']))
      echo "Saludos desde el servidor: hola {$_GET['nombre']} {$_GET['apellidos']}. ";
   else if (isset($_POST['nombre']))
      echo "Saludos desde el servidor: hola {$_POST['nombre']} {$_POST['apellidos']}. ";

   // Mostramos la fecha y hora del servidor web.
   echo "La fecha y hora del Servidor Web: ";
   echo date("j/n/Y G:i:s");
?>
```

funciones.js

```
////////////////////////////////////////////////////////
// Función cross-browser para crear objeto XMLHttpRequest
////////////////////////////////////////////////////////
function objetoXHR(){
   if (window.XMLHttpRequest){
      // El navegador implementa la interfaz XHR de forma nativa
      return new XMLHttpRequest();
   }else if (window.ActiveXObject){
      var versionesIE = new Array('Msxml2.XMLHTTP.5.0', 'Msxml2.XMLHTTP.4.0',
      'Msxml2.XMLHTTP.3.0', 'Msxml2.XMLHTTP', 'Microsoft.XMLHTTP');

      for (var i = 0; i < versionesIE.length; i++){
         try{
            /*
               Se intenta crear el objeto en Internet Explorer comenzando
               en la versión más moderna del objeto hasta la primera versión.
               En el momento que se consiga crear el objeto, saldrá del bucle
               devolviendo el nuevo objeto creado.
            */
            return new ActiveXObject(versionesIE[i]);
         }catch (errorControlado) {}//Capturamos el error,
      }
   }
```

```
   /* Si llegamos aquí es porque el navegador no posee ninguna forma de crear el objeto.
   Emitimos un mensaje de error usando el objeto Error.

   Más información sobre gestión de errores en:
   http://www.javascriptkit.com/javatutors/trycatch2.shtml */

   throw new Error("No se pudo crear el objeto XMLHttpRequest");
}


////////////////////////////////////////////////////////
// Función cross-browser para añadir Eventos
////////////////////////////////////////////////////////
var crearEvento = function(){
   function w3c_crearEvento(elemento, evento, mifuncion){
      elemento.addEventListener(evento, mifuncion, false);
   }

   function ie_crearEvento(elemento, evento, mifuncion){
      var fx = function(){
         mifuncion.call(elemento);
      };

      // Enlazamos el evento con attachEvent. Cuando usamos attachEvent
      // dejamos de tener acceso al objeto this en mifuncion. Para solucionar eso
      // usaremos el método call() del objeto Function, que nos permitirá
      // asignar el puntero this para su uso dentro de la función. El primer
      // parámetro que pongamos en call será la referencia que se usará como
      // objeto this dentro de nuestra función. De esta manera solucionamos el problema
      // de acceder a this usando attachEvent en Internet Explorer.

      elemento.attachEvent('on' + evento, fx);
   }

   if (typeof window.addEventListener !== 'undefined'){
      return w3c_crearEvento;
   }else if (typeof window.attachEvent !== 'undefined'){
      return ie_crearEvento;
   }
}();  // <= Esta es la parte más crítica - tiene que terminar en ()


////////////////////////////////////////////////////////
// Función cross-browser para modificar el contenido
// de un DIV
////////////////////////////////////////////////////////
function textoDIV(nodo, texto){
   //var nodo = document.getElementById(idObjeto);
   while (nodo.firstChild)
      nodo.removeChild(nodo.firstChild); // Eliminamos todos los hijos de ese objeto.
   // Cuando ya no tenga hijos, agregamos un hijo con el texto que recibe la función.
   nodo.appendChild(document.createTextNode(texto));
}
```

```
function iniciar(){
   // Creamos un objeto XHR.
   miXHR = new objetoXHR();

   // Cargamos de forma asíncrona el texto que nos devuelve la página
   // procesar.php
   // En este caso sólo ponemos los parámetros que pasaremos a la página procesar.php
   cargarAsync("nombre=Teresa&apellidos=Blanco Ferreiro");
}

////////////////////////////////////////////////////////
// Función cargarASync: carga el contenido con los parametros
// que se le vana a pasar a la petición AJAX de forma ASINCRONA.
////////////////////////////////////////////////////////
function cargarAsync(parametros){
   if (miXHR){
      // Activamos el indicador AJAX antes de realizar la petición.
      document.getElementById("indicador").innerHTML="<img src='AJAX-loader.gif'/>";

      // Abrimos la conexión al servidor usando método POST y a la página procesar.php
```

```
      miXHR.open('POST', "procesar.php", true);   // Abrimos la url, true=ASINCRONA

      // En cada cambio de estado(readyState) se llamará a la función estadoPeticion
      miXHR.onreadystatechange = estadoPeticion;

      // En las peticiones POST tenemos que enviar en la cabecera el Content-Type
      //ya que los datos se envían formando parte de la cabecera.
      miXHR.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

      // Hacemos la petición al servidor con los parámetros: nombre=Teresa&apellidos=Blanco...
      miXHR.send(parametros);
   }
}
```

En este ejemplo, tuvimos que realizar los siguientes cambios para adaptarlo al método **POST**:

- ✓ La función `cargarAsync()`, recibirá los parámetros por `POST` en lugar de `GET`.
- ✓ El método `.open` se modifica por:

```
miXHR.open('POST', "procesar.php", true);
```

- ✓ Tenemos que crear una cabecera con el tipo de contenido que vamos a enviar, justo antes de enviar la petición con el método `.send()`:

```
miXHR.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
```

- ✓ En el método `.send()` escribiremos los parámetros (`nombre=Teresa&apellidos=Blanco Ferreiro`) que serán enviados por el método `POST`:

```
miXHR.send(parametros.
```

2.5.- Recepción de datos en formato XML.

Cuando realizamos una petición AJAX, que nos devuelve las respuestas en formato XML, dichos datos los tendremos que consultar en la propiedad **responseXML** del objeto XHR.

index.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <title>Ejemplo dwec07 - 2.3 - AJAX ASINCRONO GET - POST</title>
      <script type="text/javascript" src="funciones.js"></script>
      <script type="text/javascript" src="index.js"></script>
      <style>
         #resultados{
            background: yellow;
         }
      </style>
   </head>
   <body>
      A continuación se cargarán por AJAX los datos recibidos en la solicitud ASINCRONA:<br/>
      Contenedor resultados:<div id="resultados"></div>
      <div id="indicador"></div>
   </body>
</html>
```

index.js

```
////////////////////////////////////////////////////////////////////
// Cuando el documento esté cargado llamamos a la función iniciar().
////////////////////////////////////////////////////////////////////
crearEvento(window,"load",iniciar);
////////////////////////////////////////////////////////

function iniciar(){
   // Creamos un objeto XHR.
   miXHR = new objetoXHR();

   // Cargamos los datos XML de forma asíncrona.
   // En este caso ponemos una página PHP que nos devuelve datos en formato XML
   // pero podríamos poner también el nombre de un fichero XML directamente: catalogos.xml
   // Se adjunta ejemplo de fichero XML.
   cargarAsync("datosxml.php");
}

////////////////////////////////////////////////////////
```

```
// Función cargarAsync: carga el contenido de la url
// usando una petición AJAX de forma ASINCRONA.
////////////////////////////////////////////////////////
function cargarAsync(url){
   if (miXHR){
      // Activamos el indicador Ajax antes de realizar la petición.
      document.getElementById("indicador").innerHTML="<img src='ajax-loader.gif'/>";

      //Si existe el objeto  miXHR
      miXHR.open('GET', url, true); //Abrimos la url, true=ASINCRONA

      // En cada cambio de estado(readyState) se llamará a la función estadoPeticion
      miXHR.onreadystatechange = estadoPeticion;

      // Hacemos la petición al servidor. Como parámetro: null ya que los datos van por GET
      miXHR.send(null);
   }
}

////////////////////////////////////////////////////////
// Función estadoPeticion: será llamada a cada cambio de estado de la petición AJAX
// cuando la respuesta del servidor es 200(fichero encontrado) y el estado de
// la solicitud es 4, accedemos a la propiedad responseText
////////////////////////////////////////////////////////
function estadoPeticion(){
   if (this.readyState==4 && this.status == 200)    {
      // Almacenamos el fichero XML en la variable datos.
      var datos=this.responseXML;

      // Tenemos que recorrer el fichero XML empleando los métodos del DOM
      // Array que contiene todos los CD's del fichero XML
      CDs= datos.documentElement.getElementsByTagName("CD");

      // En la variable salida compondremos el código HTML de la tabla a imprimir.
      salida="<table border='1'><tr><th>Titulo</th><th>Artista</th><th>Año</th></tr>";

      // Hacemos un bucle para recorrer todos los elementos CD.
      for (i=0;i<CDs.length;i++){
         salida+="<tr>";

         // Para cada CD leemos el título
         titulos=CDs[i].getElementsByTagName("TITLE");

         try{
            // Intentamos imprimir el contenido de ese elemento
            salida+="<td>" + titulos[0].firstChild.nodeValue + "</td>";
         }catch (er){
            // En el caso de que no tenga contenido ese elemento, imprimimos un espacio en
blanco
            salida+= "<td> </td>";
         }

         // Para cada CD leemos el Artista
         titulos=CDs[i].getElementsByTagName("ARTIST");
         try{
            // Intentamos imprimir el contenido de ese elemento
            salida+="<td>" + titulos[0].firstChild.nodeValue + "</td>";
         }catch (er){
            // En el caso de que no tenga contenido ese elemento, imprimimos un espacio en
blanco
            salida+="<td> </td>";
         }

         // Para cada CD leemos el Año
         titulos=CDs[i].getElementsByTagName("YEAR");
         try{
            // Intentamos imprimir el contenido de ese elemento
            salida+="<td>" + titulos[0].firstChild.nodeValue + "</td>";
         }catch (er){// En el caso de que no tenga contenido ese elemento, imprimimos un
espacio en blanco
            salida+="<td> </td>";
         }

         // Podríamos seguir sacando más campos del fichero XML..

         // Cerramos la fila.
         salida+="</tr>";
```

```
        }

        // Cuando ya no hay más Cd's cerramos la tabla.
        salida+="</table>";

        // Desactivamos el indicador AJAX cuando termina la petición
        document.getElementById("indicador").innerHTML="";

        // Imprimimos la tabla dentro del contenedor resultados.
        document.getElementById("resultados").innerHTML=salida;
    }
}
```

datosxml.php

```
<?php
    // Para que el navegador no haga cache de los datos devueltos por la página PHP.
    header('Cache-Control: no-cache, must-revalidate');
    header('Expires: Mon, 26 Jul 1997 05:00:00 GMT');

    // Leemos el contenido del fichero XML
    // e imprimimos su contenido.
    // Muy importante indicar al navegador que va a recibir contenido XML
    // eso lo hacemos con la siguiente cabecera:
    header("Content-Type: text/xml");

    $ficheroxml="<?xml version=\"1.0\" encoding=\"utf-8\"?>";
    $ficheroxml.="
    <CATALOG>
    <CD>
        <TITLE>Empire Burlesque</TITLE>
        <ARTIST>Bob Dylan</ARTIST>
        <COUNTRY>USA</COUNTRY>
        <COMPANY>Columbia</COMPANY>
        <PRICE>10.90</PRICE>
        <YEAR>1985</YEAR>
    </CD>
    <CD>
        <TITLE>Hide your heart</TITLE>
        <ARTIST>Bonnie Tyler</ARTIST>
        <COUNTRY>UK</COUNTRY>
        <COMPANY>CBS Records</COMPANY>
        <PRICE>9.90</PRICE>
        <YEAR>1988</YEAR>
    </CD>
    <CD>
        <TITLE>Greatest Hits</TITLE>
        <ARTIST>Dolly Parton</ARTIST>
        <COUNTRY>USA</COUNTRY>
        <COMPANY>RCA</COMPANY>
        <PRICE>9.90</PRICE>
        <YEAR>1982</YEAR>
    </CD>
    <CD>
        <TITLE>Still got the blues</TITLE>
        <ARTIST>Gary Moore</ARTIST>
        <COUNTRY>UK</COUNTRY>
        <COMPANY>Virgin records</COMPANY>
        <PRICE>10.20</PRICE>
        <YEAR>1990</YEAR>
    </CD>
    <CD>
        <TITLE>Eros</TITLE>
        <ARTIST>Eros Ramazzotti</ARTIST>
        <COUNTRY>EU</COUNTRY>
        <COMPANY>BMG</COMPANY>
        <PRICE>9.90</PRICE>
        <YEAR>1997</YEAR>
    </CD>
    <CD>
        <TITLE>One night only</TITLE>
        <ARTIST>Bee Gees</ARTIST>
        <COUNTRY>UK</COUNTRY>
        <COMPANY>Polydor</COMPANY>
        <PRICE>10.90</PRICE>
        <YEAR>1998</YEAR>
    </CD>
    <CD>
```

```
      <TITLE>Sylvias Mother</TITLE>
      <ARTIST></ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>CBS</COMPANY>
      <PRICE>8.10</PRICE>
      <YEAR>1973</YEAR>
   </CD>
   <CD>
      <TITLE>Maggie May</TITLE>
      <ARTIST>Rod Stewart</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>Pickwick</COMPANY>
      <PRICE>8.50</PRICE>
      <YEAR>1990</YEAR>
   </CD>
   <CD>
      <TITLE>Romanza</TITLE>
      <ARTIST>Andrea Bocelli</ARTIST>
      <COUNTRY>EU</COUNTRY>
      <COMPANY>Polydor</COMPANY>
      <PRICE>10.80</PRICE>
      <YEAR>1996</YEAR>
   </CD>
   <CD>
      <TITLE>When a man loves a woman</TITLE>
      <ARTIST>Percy Sledge</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Atlantic</COMPANY>
      <PRICE>8.70</PRICE>
      <YEAR>1987</YEAR>
   </CD>
   <CD>
      <TITLE>Black angel</TITLE>
      <ARTIST>Savage Rose</ARTIST>
      <COUNTRY>EU</COUNTRY>
      <COMPANY>Mega</COMPANY>
      <PRICE>10.90</PRICE>
      <YEAR></YEAR>
   </CD>
   <CD>
      <TITLE>1999 Grammy Nominees</TITLE>
      <ARTIST>Many</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Grammy</COMPANY>
      <PRICE>10.20</PRICE>
      <YEAR>1999</YEAR>
   </CD>
   <CD>
      <TITLE>For the good times</TITLE>
      <ARTIST>Kenny Rogers</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>Mucik Master</COMPANY>
      <PRICE>8.70</PRICE>
      <YEAR>1995</YEAR>
   </CD>
   <CD>
      <TITLE>Big Willie style</TITLE>
      <ARTIST>Will Smith</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Columbia</COMPANY>
      <PRICE>9.90</PRICE>
      <YEAR>1997</YEAR>
   </CD>
   <CD>
      <TITLE>Tupelo Honey</TITLE>
      <ARTIST>Van Morrison</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>Polydor</COMPANY>
      <PRICE>8.20</PRICE>
      <YEAR>1971</YEAR>
   </CD>
   <CD>
      <TITLE>Soulsville</TITLE>
      <ARTIST>Jorn Hoel</ARTIST>
      <COUNTRY>Norway</COUNTRY>
      <COMPANY>WEA</COMPANY>
      <PRICE>7.90</PRICE>
```

```
    <YEAR>1996</YEAR>
  </CD>
  <CD>
    <TITLE>The very best of</TITLE>
    <ARTIST>Cat Stevens</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Island</COMPANY>
    <PRICE>8.90</PRICE>
    <YEAR>1990</YEAR>
  </CD>
  <CD>
    <TITLE>Stop</TITLE>
    <ARTIST>Sam Brown</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>A and M</COMPANY>
    <PRICE>8.90</PRICE>
    <YEAR>1988</YEAR>
  </CD>
  <CD>
    <TITLE>Bridge of Spies</TITLE>
    <ARTIST>T'Pau</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Siren</COMPANY>
    <PRICE>7.90</PRICE>
    <YEAR>1987</YEAR>
  </CD>
  <CD>
    <TITLE>Private Dancer</TITLE>
    <ARTIST>Tina Turner</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Capitol</COMPANY>
    <PRICE>8.90</PRICE>
    <YEAR>1983</YEAR>
  </CD>
  <CD>
    <TITLE>Midt om natten</TITLE>
    <ARTIST>Kim Larsen</ARTIST>
    <COUNTRY>EU</COUNTRY>
    <COMPANY>Medley</COMPANY>
    <PRICE>7.80</PRICE>
    <YEAR>1983</YEAR>
  </CD>
  <CD>
    <TITLE>Pavarotti Gala Concert</TITLE>
    <ARTIST>Luciano Pavarotti</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>DECCA</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1991</YEAR>
  </CD>
  <CD>
    <TITLE>The dock of the bay</TITLE>
    <ARTIST>Otis Redding</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Atlantic</COMPANY>
    <PRICE>7.90</PRICE>
    <YEAR>1987</YEAR>
  </CD>
  <CD>
    <TITLE>Picture book</TITLE>
    <ARTIST>Simply Red</ARTIST>
    <COUNTRY>EU</COUNTRY>
    <COMPANY>Elektra</COMPANY>
    <PRICE>7.20</PRICE>
    <YEAR>1985</YEAR>
  </CD>
  <CD>
    <TITLE>Red</TITLE>
    <ARTIST>The Communards</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>London</COMPANY>
    <PRICE>7.80</PRICE>
    <YEAR>1987</YEAR>
  </CD>
  <CD>
    <TITLE>Unchain my heart</TITLE>
    <ARTIST>Joe Cocker</ARTIST>
    <COUNTRY>USA</COUNTRY>
```

```
      <COMPANY>EMI</COMPANY>
      <PRICE>8.20</PRICE>
      <YEAR>1987</YEAR>
   </CD>
</CATALOG>";

echo $ficheroxml;
?>
```

funciones.js

```
////////////////////////////////////////////////////////
// Función cross-browser para crear objeto XMLHttpRequest
////////////////////////////////////////////////////////
function objetoXHR(){
   if (window.XMLHttpRequest) {
      // El navegador implementa la interfaz XHR de forma nativa
      return new XMLHttpRequest();
   }else if (window.ActiveXObject){
      var versionesIE = new Array('Msxml2.XMLHTTP.5.0', 'Msxml2.XMLHTTP.4.0',
      'Msxml2.XMLHTTP.3.0', 'Msxml2.XMLHTTP', 'Microsoft.XMLHTTP');

      for (var i = 0; i < versionesIE.length; i++){
         try{
            /*
               Se intenta crear el objeto en Internet Explorer comenzando
               en la versión más moderna del objeto hasta la primera versión.
               En el momento que se consiga crear el objeto, saldrá del bucle
               devolviendo el nuevo objeto creado.
            */
            return new ActiveXObject(versionesIE[i]);
         }catch (errorControlado) {}//Capturamos el error,
      }
   }

   /*
      Si llegamos aquí es porque el navegador no posee ninguna forma de crear el objeto.
      Emitimos un mensaje de error usando el objeto Error.

      Más información sobre gestión de errores en:
      http://www.javascriptkit.com/javatutors/trycatch2.shtml
   */

   throw new Error("No se pudo crear el objeto XMLHttpRequest");
}


////////////////////////////////////////////////////////
// Función cross-browser para añadir Eventos
////////////////////////////////////////////////////////
var crearEvento = function(){
   function w3c_crearEvento(elemento, evento, mifuncion){
      elemento.addEventListener(evento, mifuncion, false);
   }

   function ie_crearEvento(elemento, evento, mifuncion){
      var fx = function(){
         mifuncion.call(elemento);
      };

      // Enlazamos el evento con attachEvent. Cuando usamos attachEvent
      // dejamos de tener acceso al objeto this en mifuncion. Para solucionar eso
      // usaremos el método call() del objeto Function, que nos permitirá
      // asignar el puntero this para su uso dentro de la función. El primer
      // parámetro que pongamos en call será la referencia que se usará como
      // objeto this dentro de nuestra función. De esta manera solucionamos el problema
      // de acceder a this usando attachEvent en Internet Explorer.

      elemento.attachEvent('on' + evento, fx);
   }

   if (typeof window.addEventListener !== 'undefined'){
      return w3c_crearEvento;
   }else if (typeof window.attachEvent !== 'undefined'){
      return ie_crearEvento;
   }
}();  // <= Esta es la parte más crítica - tiene que terminar en ()
```

```
////////////////////////////////////////////////////
// Función cross-browser para modificar el contenido
// de un DIV
////////////////////////////////////////////////////
function textoDIV(nodo, texto){
   //var nodo = document.getElementById(idObjeto);
   while (nodo.firstChild)
      nodo.removeChild(nodo.firstChild); // Eliminamos todos los hijos de ese objeto.
   // Cuando ya no tenga hijos, agregamos un hijo con el texto que recibe la función.
   nodo.appendChild(document.createTextNode(texto));
}
```

catalogo.xml

```
<?xml version="1.0" encoding="utf-8"?>
<CATALOG>
   <CD>
      <TITLE>Empire Burlesque</TITLE>
      <ARTIST>Bob Dylan</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Columbia</COMPANY>
      <PRICE>10.90</PRICE>
      <YEAR>1985</YEAR>
   </CD>
   <CD>
      <TITLE>Hide your heart</TITLE>
      <ARTIST>Bonnie Tyler</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>CBS Records</COMPANY>
      <PRICE>9.90</PRICE>
      <YEAR>1988</YEAR>
   </CD>
   <CD>
      <TITLE>Greatest Hits</TITLE>
      <ARTIST>Dolly Parton</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>RCA</COMPANY>
      <PRICE>9.90</PRICE>
      <YEAR>1982</YEAR>
   </CD>
   <CD>
      <TITLE>Still got the blues</TITLE>
      <ARTIST>Gary Moore</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>Virgin records</COMPANY>
      <PRICE>10.20</PRICE>
      <YEAR>1990</YEAR>
   </CD>
   <CD>
      <TITLE>Eros</TITLE>
      <ARTIST>Eros Ramazzotti</ARTIST>
      <COUNTRY>EU</COUNTRY>
      <COMPANY>BMG</COMPANY>
      <PRICE>9.90</PRICE>
      <YEAR>1997</YEAR>
   </CD>
   <CD>
      <TITLE>One night only</TITLE>
      <ARTIST>Bee Gees</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>Polydor</COMPANY>
      <PRICE>10.90</PRICE>
      <YEAR>1998</YEAR>
   </CD>
   <CD>
      <TITLE>Sylvias Mother</TITLE>
      <ARTIST>Dr.Hook</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>CBS</COMPANY>
      <PRICE>8.10</PRICE>
      <YEAR>1973</YEAR>
   </CD>
   <CD>
      <TITLE>Maggie May</TITLE>
      <ARTIST>Rod Stewart</ARTIST>
      <COUNTRY>UK</COUNTRY>
```

```
    <COMPANY>Pickwick</COMPANY>
    <PRICE>8.50</PRICE>
    <YEAR>1990</YEAR>
  </CD>
  <CD>
    <TITLE>Romanza</TITLE>
    <ARTIST>Andrea Bocelli</ARTIST>
    <COUNTRY>EU</COUNTRY>
    <COMPANY>Polydor</COMPANY>
    <PRICE>10.80</PRICE>
    <YEAR>1996</YEAR>
  </CD>
  <CD>
    <TITLE>When a man loves a woman</TITLE>
    <ARTIST>Percy Sledge</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Atlantic</COMPANY>
    <PRICE>8.70</PRICE>
    <YEAR>1987</YEAR>
  </CD>
  <CD>
    <TITLE>Black angel</TITLE>
    <ARTIST>Savage Rose</ARTIST>
    <COUNTRY>EU</COUNTRY>
    <COMPANY>Mega</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1995</YEAR>
  </CD>
  <CD>
    <TITLE>1999 Grammy Nominees</TITLE>
    <ARTIST>Many</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Grammy</COMPANY>
    <PRICE>10.20</PRICE>
    <YEAR>1999</YEAR>
  </CD>
  <CD>
    <TITLE>For the good times</TITLE>
    <ARTIST>Kenny Rogers</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Mucik Master</COMPANY>
    <PRICE>8.70</PRICE>
    <YEAR>1995</YEAR>
  </CD>
  <CD>
    <TITLE>Big Willie style</TITLE>
    <ARTIST>Will Smith</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1997</YEAR>
  </CD>
  <CD>
    <TITLE>Tupelo Honey</TITLE>
    <ARTIST>Van Morrison</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Polydor</COMPANY>
    <PRICE>8.20</PRICE>
    <YEAR>1971</YEAR>
  </CD>
  <CD>
    <TITLE>Soulsville</TITLE>
    <ARTIST>Jorn Hoel</ARTIST>
    <COUNTRY>Norway</COUNTRY>
    <COMPANY>WEA</COMPANY>
    <PRICE>7.90</PRICE>
    <YEAR>1996</YEAR>
  </CD>
  <CD>
    <TITLE>The very best of</TITLE>
    <ARTIST>Cat Stevens</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Island</COMPANY>
    <PRICE>8.90</PRICE>
    <YEAR>1990</YEAR>
  </CD>
  <CD>
```

```
      <TITLE>Stop</TITLE>
      <ARTIST>Sam Brown</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>A and M</COMPANY>
      <PRICE>8.90</PRICE>
      <YEAR>1988</YEAR>
   </CD>
   <CD>
      <TITLE>Bridge of Spies</TITLE>
      <ARTIST>T'Pau</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>Siren</COMPANY>
      <PRICE>7.90</PRICE>
      <YEAR>1987</YEAR>
   </CD>
   <CD>
      <TITLE>Private Dancer</TITLE>
      <ARTIST>Tina Turner</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>Capitol</COMPANY>
      <PRICE>8.90</PRICE>
      <YEAR>1983</YEAR>
   </CD>
   <CD>
      <TITLE>Midt om natten</TITLE>
      <ARTIST>Kim Larsen</ARTIST>
      <COUNTRY>EU</COUNTRY>
      <COMPANY>Medley</COMPANY>
      <PRICE>7.80</PRICE>
      <YEAR>1983</YEAR>
   </CD>
   <CD>
      <TITLE>Pavarotti Gala Concert</TITLE>
      <ARTIST>Luciano Pavarotti</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>DECCA</COMPANY>
      <PRICE>9.90</PRICE>
      <YEAR>1991</YEAR>
   </CD>
   <CD>
      <TITLE>The dock of the bay</TITLE>
      <ARTIST>Otis Redding</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Atlantic</COMPANY>
      <PRICE>7.90</PRICE>
      <YEAR>1987</YEAR>
   </CD>
   <CD>
      <TITLE>Picture book</TITLE>
      <ARTIST>Simply Red</ARTIST>
      <COUNTRY>EU</COUNTRY>
      <COMPANY>Elektra</COMPANY>
      <PRICE>7.20</PRICE>
      <YEAR>1985</YEAR>
   </CD>
   <CD>
      <TITLE>Red</TITLE>
      <ARTIST>The Communards</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>London</COMPANY>
      <PRICE>7.80</PRICE>
      <YEAR>1987</YEAR>
   </CD>
   <CD>
      <TITLE>Unchain my heart</TITLE>
      <ARTIST>Joe Cocker</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>EMI</COMPANY>
      <PRICE>8.20</PRICE>
      <YEAR>1987</YEAR>
   </CD>
</CATALOG>
```

En la función **`iniciar()`**, le hemos dicho que cargue de forma asíncrona, empleando el método **`GET`**, el fichero `datosXML.php`. Esta aplicación PHP, nos devolverá un fichero XML, con una lista de Cd de música con el artista, país, compañía, etc.

Instrucción de carga del fichero datosXML.php: `cargarAsync("datosXML.php");`

Si queremos cargar directamente un fichero XML, y conocemos su nombre, podremos escribir directamente:

```
cargarAsync("catalogo.XML");
```

En la función de `estadoPeticion()`, cuando `readyState` es 4 y el `status` es **OK** (200), accedemos a los resultados de la petición AJAX, en la propiedad `responseXML`. Para gestionar los datos XML, tendremos que recorrerlos empleando los métodos del DOM, ya que un fichero XML comparte la estructura en árbol de un documento HTML, y podemos utilizar, por tanto, los mismos métodos que empleamos para recorrer el DOM HTML.

En nuestro caso, lo primero que vamos a hacer es recorrer los elementos, que son los que contienen toda la información referente a los cd's de música:

```
// Almacenamos el fichero XML en la variable resultados.
resultados=this.responseXML;

// Tenemos que recorrer el fichero XML empleando los métodos del DOM
// Array que contiene todos los CD's del fichero XML
CDs= resultados.documentElement.getElementsByTagName("CD");
```

Haremos un bucle para recorrer todos los cd's del catálogo, y dentro de cada uno, imprimiremos los datos que nos interesen:

```
// Hacemos un bucle para recorrer todos los elementos CD.
for (i=0;i<CDs.length;i++){
…..
```

Dentro de cada CD, accederemos al elemento que nos interese e imprimiremos su contenido. Para imprimir el contenido de cada nodo, tendremos que hacerlo con el comando `try { } catch {}`, ya que si intentamos acceder a un nodo que no tenga contenido, nos dará un error de JavaScript, puesto que el elemento hijo no existe, y entonces se detendrá la ejecución de JavaScript y no imprimirá nuestro listado.

```
// Para cada CD leemos el título
titulos=CDs[i].getElementsByTagName("TITLE");

try{
   // Intentamos acceder al contenido de ese elemento
   salida+="<td>" + titulos[0].firstChild.nodeValue + "</td>";
}catch (er){
   // En el caso de que no tenga contenido ese elemento imprimimos un espacio en blanco.
   salida+= "<td> </td>";
}
```

2.6.- Recepción de datos en formato JSON (parte I).

Otro formato de intercambio muy utilizado en AJAX, es el formato JSON. JSON es un formato de intercambio de datos, alternativo a XML, mucho más simple de leer, escribir e interpretar. Significa **JavaScript Object Notation**, y consiste en escribir los datos en formato de Javascript. Vamos a hacer un poco de repaso:

Arrays

Se pueden crear con corchetes:

```
var Beatles = ["Paul","John","George","Ringo"];
```

Con `new Array()`:

```
var Beatles = new Array("Paul","John","George","Ringo");
```

O también de la siguiente forma:

```
var Beatles = { "Paul","John","George","Ringo");
```

Objetos

Un objeto literal se puede crear entre llaves: `{ propiedad1:valor, propiedad2:valor, propiedad3:valor}`

```
var Beatles = {
 "Country" : "England",
 "YearFormed" : 1959,
 "Style" : "Rock'n'Roll"
}
```

Que será equivalente a:

```
var Beatles = new Object();

Beatles.Country = "England";
Beatles.YearFormed = 1959;
Beatles.Style = "Rock'n'Roll";
```

Y para acceder a sus propiedades lo podemos hacer con:

```
alert(Beatles.Style); // Notación de puntos
alert(Beatles["Style"]); // Notación de corchetes
```

Los objetos también pueden contener arrays literales: [...]

```
var Beatles = {
 "Country" : "England",
 "YearFormed" : 1959,
 "Style" : "Rock'n'Roll",
 "Members" : ["Paul","John","George","Ringo"]
}
```

Y los arrays literales podrán contener objetos literales a su vez: {…} , {...}

```
var Rockbands =
[
{  "Name" : "Beatles",  "Country" : "England",  "YearFormed" : 1959,  "Style" : "Rock'n'Roll",
"Members" :
 ["Paul","John","George","Ringo"] } , {  "Name" : "Rolling Stones",  "Country" : :England",
 "YearFormed" : 1962,  "Style" : "Rock'n'Roll", "Members" : ["Mick","Keith","Charlie","Bill"]
}
]
```

La sintaxis de JSON es como la sintaxis literal de un objeto, excepto que, esos objetos no pueden ser asignados a una variable. JSON representará los datos en sí mismos. Por lo tanto el objeto *Beatles* que vimos antes se definiría de la siguiente forma:

```
{
 "Name" : "Beatles",
 "Country" : "England",
 "YearFormed" : 1959,
 "Style" : "Rock'n'Roll",
 "Members" : ["Paul","John","George","Ringo"]
}
```

Una cadena JSON es, simplemente, una cadena de texto, y no un objeto en sí misma. Necesita ser convertida a un objeto antes de poder ser utilizada en JavaScript. Ésto se puede hacer con la función `eval()` de JavaScript y también se pueden usar lo que se conoce como analizadores JSON, que facilitarán esa conversión.

2.7.- Recepción de datos en formato JSON (parte II).

Veamos un ejemplo completo en el que recibimos, con AJAX, los datos procedentes de un listado de una tabla MySQL en formato JSON.

index.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
```

```
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <title>Ejemplo dwec07 - 2.7 - JSON Ajax asíncrono</title>
      <script type="text/javascript" src="funciones.js"></script>
      <script type="text/javascript" src="index.js"></script>
      <style>
         #resultados{
            background: yellow;
         }
      </style>
   </head>
   <body>
      A continuación se cargarán por AJAX los datos recibidos en la solicitud ASINCRONA:<br/>
      Contenedor resultados:<div id="resultados"></div>
      <div id="indicador"></div>
   </body>
</html>
```

index.js

```
////////////////////////////////////////////////////////////////////
// Cuando el documento esté cargado llamamos a la función iniciar().
////////////////////////////////////////////////////////////////////
crearEvento(window,"load",iniciar);
////////////////////////////////////////////////////////


function iniciar(){
   // Creamos un objeto XHR.
   miXHR = new objetoXHR();

   // Cargamos los datos XML de forma asíncrona.
   // En este caso ponemos una página PHP que nos devuelve datos en formato XML
   // pero podríamos poner también el nombre de un fichero XML directamente: catalogos.xml
   // Se adjunta ejemplo de fichero XML.
   cargarAsync("datosjson.php");
}

////////////////////////////////////////////////////////
// Función cargarAsync: carga el contenido de la url
// usando una petición AJAX de forma ASINCRONA.
////////////////////////////////////////////////////////
function cargarAsync(url){
   if (miXHR){
      // Activamos el indicador Ajax antes de realizar la petición.
      document.getElementById("indicador").innerHTML="<img src='ajax-loader.gif'/>";

      //Si existe el objeto  miXHR
      miXHR.open('GET', url, true); //Abrimos la url, true=ASINCRONA

      // En cada cambio de estado(readyState) se llamará a la función estadoPeticion
      miXHR.onreadystatechange = estadoPeticion;

      // Hacemos la petición al servidor. Como parámetro: null ya que los datos van por GET
      miXHR.send(null);
   }
}

////////////////////////////////////////////////////////
// Función estadoPeticion: será llamada a cada cambio de estado de la petición AJAX
// cuando la respuesta del servidor es 200(fichero encontrado) y el estado de
// la solicitud es 4, accedemos a la propiedad responseText
////////////////////////////////////////////////////////
function estadoPeticion(){
   if (this.readyState==4 && this.status == 200){
      // Los datos JSON los recibiremos como texto en la propiedad this.responseText

      // Con la función eval() evaluaremos ese resultado para convertir a objetos y variables
el string que estamos recibiendo en JSON.
      // Lo que recibimos en formato JSON es un string que representa un array [ ... ] que
contiene objetos literales {...},{...},...
      /*            Ejemplo:           [            {"id":"2","nombrecentro":"IES            A
Piringalla","localidad":"Lugo","provincia":"Lugo","telefono":"982212010","fechavisita":"2010-
11-26","numvisitantes":"85"} , {"id":"10","nombrecentro":"IES As Fontiñas","localidad" : .....
} ]  */

      // Asignamos a la variable resultados la evaluación de responseText
```

```
      var resultados=eval( '(' +this.responseText+')');

      texto                  =                  "<table                  border=1><tr><th>Nombre
Centro</th><th>Localidad</th><th>Provincia</th><th>Telefono</th><th>Fecha
Visita</th><th>Numero Visitantes</th></tr>";
      // Hacemos un bucle para recorrer todos los objetos literales recibidos en el array
resultados y mostrar su contenido.
      for (var i=0; i < resultados.length; i++){
         objeto = resultados[i];
         texto+="<tr><td>"+objeto.nombrecentro+"</td><td>"+
            objeto.localidad+"</td><td>"+objeto.provincia+"</td><td>"+
            objeto.telefono+"</td><td>"+objeto.fechavisita+"</td><td>"+
            objeto.numvisitantes+"</td></tr>";
      }

      // Desactivamos el indicador AJAX cuando termina la petición
      document.getElementById("indicador").innerHTML="";

      // Imprimimos la tabla dentro del contenedor resultados.
      document.getElementById("resultados").innerHTML=texto;
   }
}
```

funciones.js

```
////////////////////////////////////////////////////////
// Función cross-browser para crear objeto XMLHttpRequest
////////////////////////////////////////////////////////
function objetoXHR(){
   if (window.XMLHttpRequest){
      // El navegador implementa la interfaz XHR de forma nativa
      return new XMLHttpRequest();
   }else if (window.ActiveXObject){
      var versionesIE = new Array('Msxml2.XMLHTTP.5.0', 'Msxml2.XMLHTTP.4.0',
      'Msxml2.XMLHTTP.3.0', 'Msxml2.XMLHTTP', 'Microsoft.XMLHTTP');

      for (var i = 0; i < versionesIE.length; i++){
         try{
            /*
               Se intenta crear el objeto en Internet Explorer comenzando
               en la versión más moderna del objeto hasta la primera versión.
               En el momento que se consiga crear el objeto, saldrá del bucle
               devolviendo el nuevo objeto creado.
            */
            return new ActiveXObject(versionesIE[i]);
         }catch (errorControlado) {}//Capturamos el error,
      }
   }

   /*
      Si llegamos aquí es porque el navegador no posee ninguna forma de crear el objeto.
      Emitimos un mensaje de error usando el objeto Error.

      Más información sobre gestión de errores en:
      http://www.javascriptkit.com/javatutors/trycatch2.shtml
   */

   throw new Error("No se pudo crear el objeto XMLHttpRequest");
}


////////////////////////////////////////////////////////
// Función cross-browser para añadir Eventos
////////////////////////////////////////////////////////
var crearEvento = function(){
   function w3c_crearEvento(elemento, evento, mifuncion){
      elemento.addEventListener(evento, mifuncion, false);
   }

   function ie_crearEvento(elemento, evento, mifuncion){
      var fx = function(){
         mifuncion.call(elemento);
      };

      // Enlazamos el evento con attachEvent. Cuando usamos attachEvent
      // dejamos de tener acceso al objeto this en mifuncion. Para solucionar eso
      // usaremos el método call() del objeto Function, que nos permitirá
      // asignar el puntero this para su uso dentro de la función. El primer
```

```
      // parámetro que pongamos en call será la referencia que se usará como
      // objeto this dentro de nuestra función. De esta manera solucionamos el problema
      // de acceder a this usando attachEvent en Internet Explorer.

      elemento.attachEvent('on' + evento, fx);
   }

   if (typeof window.addEventListener !== 'undefined'){
      return w3c_crearEvento;
   }else if (typeof window.attachEvent !== 'undefined'){
      return ie_crearEvento;
   }
}();  // <= Esta es la parte más crítica - tiene que terminar en ()


////////////////////////////////////////////////////
// Función cross-browser para modificar el contenido
// de un DIV
////////////////////////////////////////////////////
function textoDIV(nodo, texto){
   //var nodo = document.getElementById(idObjeto);
   while (nodo.firstChild)
      nodo.removeChild(nodo.firstChild); // Eliminamos todos los hijos de ese objeto.
   // Cuando ya no tenga hijos, agregamos un hijo con el texto que recibe la función.
   nodo.appendChild(document.createTextNode(texto));
}
```

datosjson.php

```
<?php
   // Cabecera para indicar que vamos a enviar datos JSON y que no haga caché de los datos.
   header('Content-Type: application/json');
   header('Cache-Control: no-cache, must-revalidate');
   header('Expires: Mon, 26 Jul 1997 05:00:00 GMT');

   /*
      Utilizar el fichero dbcreacion.sql incluído en la carpeta para crear la base de datos,
      usuario y tabla en tu servidor MySQL.
      Si fuera necesario modifica los datos de la configuración y adáptalos a tu entorno
      de trabajo.
   */

   // Configuración BASE DE DATOS MYSQL
   $servidor = "localhost";
   $basedatos = "ajax";
   $usuario = "ajax";
   $password = "dwec";

   // Creamos la conexión al servidor.
   $conexion=mysql_connect($servidor, $usuario, $password) or die(mysql_error());
   mysql_query("SET NAMES 'utf8'",$conexion);

   // Seleccionar la base de datos en esa conexion.
   mysql_select_db($basedatos,$conexion) or die(mysql_error());

   // Consulta SQL para obtener los datos de los centros.
   $sql="select * from centros order by nombrecentro";
   $resultados=mysql_query($sql,$conexion) or die(mysql_error());

   while ( $fila = mysql_fetch_array($resultados, MYSQL_ASSOC))
   {
      // Almacenamos en un array  cada una de las filas que vamos leyendo del recordset.
      $datos[]=$fila;
   }

   // Como resultado se puede enviar algo similar a:
   /*
      [      {"id":"3","nombrecentro":"IES      San      Clemente","localidad":"Santiago      de
Compostela","provincia":"A      Coruña"      ,"telefono":"981580496","fechavisita":"2010-11-26",
"numvisitantes":"60"} , {"id":"10","nombrecentro":"IES As Fontiñas","localidad" : ..... } ]

      // Empleando la siguiente instrucción:
      echo json_encode($datos);
   */

   /*
      O si queremos enviar como resultado un array de objetos literales llamado
```

```
      resultados, haremos:
      resultados               =               [{"id":"2","nombrecentro":"IES               A
Piringalla","localidad":"Lugo","provincia":"Lugo","telefono":"982212010","fechavisita":"2010-
11-26","numvisitantes":"85"} , {"id":"10","nombrecentro":"IES As Fontiñas","localidad" : .....
}]

      Empleando la siguiente instrucción:
      echo "resultados=".json_encode($datos);

      De esta forma en la página index.js sólo tendríamos que poner
      eval('(' + this.responseText + ')')
      y así ya tenemos disponible en JavaScript una variable resultados que es un array
      que contendrá los objetos literales.
   */

   echo json_encode($datos); // función de PHP que convierte a formato JSON el array.

   mysql_close($conexion);
?>
```

dbcreacion.sql

```
CREATE USER 'ajax'@'localhost' IDENTIFIED BY 'dwec';

GRANT USAGE ON * . * TO 'ajax'@'localhost' IDENTIFIED BY 'dwec' WITH MAX_QUERIES_PER_HOUR 0
MAX_CONNECTIONS_PER_HOUR 0 MAX_UPDATES_PER_HOUR 0 MAX_USER_CONNECTIONS 0 ;

CREATE DATABASE IF NOT EXISTS `ajax` ;

GRANT ALL PRIVILEGES ON `ajax` . * TO 'ajax'@'localhost';

use  `ajax`;


CREATE TABLE IF NOT EXISTS `centros` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `nombrecentro` varchar(150) NOT NULL,
  `localidad` varchar(100) NOT NULL,
  `provincia` varchar(50) NOT NULL,
  `telefono` varchar(9) NOT NULL,
  `fechavisita` date NOT NULL,
  `numvisitantes` int(10) unsigned NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=MyISAM  DEFAULT CHARSET=latin1 AUTO_INCREMENT=11 ;

--
-- Volcar la base de datos para la tabla `centros`
--

INSERT  INTO  `centros`  (`id`,  `nombrecentro`,  `localidad`,  `provincia`,  `telefono`,
`fechavisita`, `numvisitantes`) VALUES
(1, 'IES Ramon Mª Aller Ulloa', 'Lalin', 'Pontevedra', '986780114', '2010-11-26', 90),
(2, 'IES A Piringalla', 'Lugo', 'Lugo', '982212010', '2010-11-26', 85),
(3, 'IES San Clemente', 'Santiago de Compostela', 'A Coruña', '981580496', '2010-11-26', 60),
(4, 'IES de Teis', 'Vigo', 'Pontevedra', '986373811', '2010-11-27', 72),
(5, 'IES Leliadoura', 'Ribeira', 'A Coruña', '981874633', '2010-11-25', 0),
(6, 'IES Cruceiro Baleares', 'Culleredo', 'A Coruña', '981660700', '2010-11-26', 30),
(7, 'IES Leliadoura', 'Ribeira', 'A Coruña', '981874633', '2010-11-25', 50),
(8, 'IES Cruceiro Baleares', 'Culleredo', 'A Coruña', '981660700', '2010-11-26', 30),
(9, 'IES As Lagoas', 'Ourense', 'Ourense', '988391325', '2010-11-26', 35),
(10, 'IES As Fontiñas', 'Santiago de Compostela', 'A Coruña', '981573440', '2010-11-27', 64);
```

catalogo.xml

```
<?xml version="1.0" encoding="utf-8"?>
<CATALOG>
   <CD>
      <TITLE>Empire Burlesque</TITLE>
      <ARTIST>Bob Dylan</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Columbia</COMPANY>
      <PRICE>10.90</PRICE>
      <YEAR>1985</YEAR>
   </CD>
   <CD>
      <TITLE>Hide your heart</TITLE>
      <ARTIST>Bonnie Tyler</ARTIST>
      <COUNTRY>UK</COUNTRY>
```

```
    <COMPANY>CBS Records</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1988</YEAR>
</CD>
<CD>
    <TITLE>Greatest Hits</TITLE>
    <ARTIST>Dolly Parton</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>RCA</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1982</YEAR>
</CD>
<CD>
    <TITLE>Still got the blues</TITLE>
    <ARTIST>Gary Moore</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Virgin records</COMPANY>
    <PRICE>10.20</PRICE>
    <YEAR>1990</YEAR>
</CD>
<CD>
    <TITLE>Eros</TITLE>
    <ARTIST>Eros Ramazzotti</ARTIST>
    <COUNTRY>EU</COUNTRY>
    <COMPANY>BMG</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1997</YEAR>
</CD>
<CD>
    <TITLE>One night only</TITLE>
    <ARTIST>Bee Gees</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Polydor</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1998</YEAR>
</CD>
<CD>
    <TITLE>Sylvias Mother</TITLE>
    <ARTIST>Dr.Hook</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>CBS</COMPANY>
    <PRICE>8.10</PRICE>
    <YEAR>1973</YEAR>
</CD>
<CD>
    <TITLE>Maggie May</TITLE>
    <ARTIST>Rod Stewart</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Pickwick</COMPANY>
    <PRICE>8.50</PRICE>
    <YEAR>1990</YEAR>
</CD>
<CD>
    <TITLE>Romanza</TITLE>
    <ARTIST>Andrea Bocelli</ARTIST>
    <COUNTRY>EU</COUNTRY>
    <COMPANY>Polydor</COMPANY>
    <PRICE>10.80</PRICE>
    <YEAR>1996</YEAR>
</CD>
<CD>
    <TITLE>When a man loves a woman</TITLE>
    <ARTIST>Percy Sledge</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Atlantic</COMPANY>
    <PRICE>8.70</PRICE>
    <YEAR>1987</YEAR>
</CD>
<CD>
    <TITLE>Black angel</TITLE>
    <ARTIST>Savage Rose</ARTIST>
    <COUNTRY>EU</COUNTRY>
    <COMPANY>Mega</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1995</YEAR>
</CD>
<CD>
```

```
    <TITLE>1999 Grammy Nominees</TITLE>
    <ARTIST>Many</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Grammy</COMPANY>
    <PRICE>10.20</PRICE>
    <YEAR>1999</YEAR>
  </CD>
  <CD>
    <TITLE>For the good times</TITLE>
    <ARTIST>Kenny Rogers</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Mucik Master</COMPANY>
    <PRICE>8.70</PRICE>
    <YEAR>1995</YEAR>
  </CD>
  <CD>
    <TITLE>Big Willie style</TITLE>
    <ARTIST>Will Smith</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1997</YEAR>
  </CD>
  <CD>
    <TITLE>Tupelo Honey</TITLE>
    <ARTIST>Van Morrison</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Polydor</COMPANY>
    <PRICE>8.20</PRICE>
    <YEAR>1971</YEAR>
  </CD>
  <CD>
    <TITLE>Soulsville</TITLE>
    <ARTIST>Jorn Hoel</ARTIST>
    <COUNTRY>Norway</COUNTRY>
    <COMPANY>WEA</COMPANY>
    <PRICE>7.90</PRICE>
    <YEAR>1996</YEAR>
  </CD>
  <CD>
    <TITLE>The very best of</TITLE>
    <ARTIST>Cat Stevens</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Island</COMPANY>
    <PRICE>8.90</PRICE>
    <YEAR>1990</YEAR>
  </CD>
  <CD>
    <TITLE>Stop</TITLE>
    <ARTIST>Sam Brown</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>A and M</COMPANY>
    <PRICE>8.90</PRICE>
    <YEAR>1988</YEAR>
  </CD>
  <CD>
    <TITLE>Bridge of Spies</TITLE>
    <ARTIST>T'Pau</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Siren</COMPANY>
    <PRICE>7.90</PRICE>
    <YEAR>1987</YEAR>
  </CD>
  <CD>
    <TITLE>Private Dancer</TITLE>
    <ARTIST>Tina Turner</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Capitol</COMPANY>
    <PRICE>8.90</PRICE>
    <YEAR>1983</YEAR>
  </CD>
  <CD>
    <TITLE>Midt om natten</TITLE>
    <ARTIST>Kim Larsen</ARTIST>
    <COUNTRY>EU</COUNTRY>
    <COMPANY>Medley</COMPANY>
    <PRICE>7.80</PRICE>
    <YEAR>1983</YEAR>
```

```
   </CD>
   <CD>
      <TITLE>Pavarotti Gala Concert</TITLE>
      <ARTIST>Luciano Pavarotti</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>DECCA</COMPANY>
      <PRICE>9.90</PRICE>
      <YEAR>1991</YEAR>
   </CD>
   <CD>
      <TITLE>The dock of the bay</TITLE>
      <ARTIST>Otis Redding</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Atlantic</COMPANY>
      <PRICE>7.90</PRICE>
      <YEAR>1987</YEAR>
   </CD>
   <CD>
      <TITLE>Picture book</TITLE>
      <ARTIST>Simply Red</ARTIST>
      <COUNTRY>EU</COUNTRY>
      <COMPANY>Elektra</COMPANY>
      <PRICE>7.20</PRICE>
      <YEAR>1985</YEAR>
   </CD>
   <CD>
      <TITLE>Red</TITLE>
      <ARTIST>The Communards</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>London</COMPANY>
      <PRICE>7.80</PRICE>
      <YEAR>1987</YEAR>
   </CD>
   <CD>
      <TITLE>Unchain my heart</TITLE>
      <ARTIST>Joe Cocker</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>EMI</COMPANY>
      <PRICE>8.20</PRICE>
      <YEAR>1987</YEAR>
   </CD>
</CATALOG>
```

Si quieres probar el ejemplo, necesitas un servidor web, PHP y MySQL. Puedes instalarte por ejemplo XAMPP en cualquiera de sus versiones para Windows o Linux.

En este ejemplo, se realiza una consulta a una tabla (se adjunta código SQL de creación de la base de datos, usuario, contraseña y la tabla con los datos). La página PHP, devolverá los resultados en una cadena de texto, que contiene un array de objetos literales en formato JSON.

Código de PHP del fichero `datosJSON.php`:

```
….
// Consulta SQL para obtener los datos de los centros.
$sql="select * from centros order by nombrecentro";
$resultados=mysql_query($sql,$conexion) or die(mysql_error());

while ( $fila = mysql_fetch_array($resultados, MYSQL_ASSOC)){
   // Almacenamos en un array  cada una de las filas que vamos leyendo del recordset.
   $datos[]=$fila;
}

echo JSON_encode($datos); // función de PHP que convierte a formato JSON el array.
….
```

Nuestra aplicación de JavaScript recibe, por AJAX, esos datos, en la propiedad **`responseText`**. Para poder utilizar directamente esos datos en JavaScript, lo que tenemos que hacer es evaluar la expresión (cadena de texto, que recibimos de la página `datosJSON.php`). La expresión JSON contenía

un array de objetos literales, por lo que tenemos que crear una variable, para asignar ese array y poder recorrerlo.

Para evaluar la expresión lo hacemos con la función `eval()` de JavaScript:

```
// Asignamos a la variable resultados la evaluación de responseText
var resultados=eval( '(' +this.responseText+')');

// Hacemos un bucle para recorrer todos los objetos literales recibidos en el array resultados
y mostrar su contenido.
for (var i=0; i < resultados.length; i++){
     objeto = resultados[i];
     texto+="<tr><td>"+objeto.nombrecentro+
         "</td><td>"+objeto.localidad+"</td><td>"+
         objeto.provincia+"</td><td>"+
         objeto.telefono+"</td><td>"+
         objeto.fechavisita +"</td><td>"+
         objeto.numvisitantes+ "</td></tr>";
}
```

"Si vivir es durar, prefiero una canción de los Beatles a un Long Play de los Boston Pops. - Mafalda."
Quino, Joaquín Salvador Lavado

3.- Librerías cross-browser para programación AJAX.

Caso práctico

*El estudio del objeto para trabajar con AJAX, está comenzando a dar sus frutos. Lo que más le fastidia a **Antonio**, es que necesita programar bastante código, y aunque puede crear alguna librería para acelerar la programación, también ve que las diferentes incompatibilidades, entre navegadores, no van a ayudar nada en esta labor. Por esta razón está un poco desilusionado, por que le va a suponer bastante trabajo, aunque los resultados merecen la pena.*

*En ese momento llega **Juan**, y le da la sorpresa que le había comentado hace unos días. Le facilita un pequeño tutorial sobre la librería jQuery, que le va a permitir hacer peticiones AJAX utilizando prácticamente una línea de código. No tendrá que preocuparse por temas de cross-browsing y, además, la misma librería le facilitará métodos para hacer todo tipo de efectos, animaciones, etc. Esta librería cuenta además con infinidad de complementos gratuitos, que permiten hacer prácticamente cualquier cosa en muy poco tiempo.*

La programación con AJAX, es uno de los pilares de lo que se conoce como web 2.0, término que incluye a las aplicaciones web que facilitan el compartir información, la interoperabilidad, el diseño centrado en el usuario y la colaboración web. Ejemplos de la web 2.0, pueden ser las comunidades web, los servicios web, aplicaciones web, redes sociales, servicios de alojamiento de vídeos, wikis, blogs, mashup (*página web o aplicación que usa y combina datos, presentaciones y funcionalidad procedentes de una o más fuentes para crear nuevos servicios. El término implica integración fácil y rápida, usando a menudo APIs abiertos y fuentes de datos con el objetivo de producir resultados enriquecidos combinando diferentes fuentes*), etc.

Más información sobre Mashup (aplicaciones web híbridas).

http://es.wikipedia.org/wiki/Mashup_%28aplicaci%C3%B3n_web_h%C3%ADbrida%29

Gracias a las aplicaciones web 2.0, se han desarrollado gran cantidad de utilidades/herramientas/frameworks para el desarrollo web con JavaScript, DHTML(HTML dinámico) y AJAX. La gran ventaja de usar alguna librería o framework para AJAX, es la del ahorro de tiempo y código, en nuestras aplicaciones. Veremos que con algunas librerías vamos a realizar peticiones AJAX, con una simple instrucción de código sin tener que preocuparnos de crear el objeto XHR, ni gestionar el código de respuesta del servidor, los estados de la solicitud, etc.

Otra de las ventajas que nos aportan este tipo de librerías, es la de la compatibilidad entre navegadores (cross-browser). De esta forma tenemos un problema menos, ya que la propia librería será capaz de crear la petición AJAX de una forma u otra, dependiendo del navegador que estemos utilizando.

A principios del año 2008 Google liberó su API de librerías AJAX, como una red de distribución de contenido y arquitectura de carga, para algunos de los frameworks más populares. Mediante esta API se eliminan las dificultades a la hora de desarrollar mashups en JavaScript. Se elimina el problema de alojar las librerías (ya que están centralizadas en Google), configurar las cabeceras de cache, etc. Esta API ofrece acceso a las siguientes librerías Open Source, realizadas con JavaScript:

- ✓ jQuery.
- ✓ prototype.
- ✓ scriptaculous.
- ✓ mooTools.
- ✓ dojo, swfobject, chrome-frame, webfont, etc.

Los scripts de estas librerías están accesibles directamente utilizando la URL de descarga, o a través del método *google.load*() del cargador de la API AJAX de Google.

Hay muchísimas librerías que se pueden utilizar para programar AJAX, dependiendo del lenguaje que utilicemos. Mira el siguiente anexo para comprobarlo

Nosotros nos vamos a centrar en el uso de la librería jQuery, por ser una de las más utilizadas hoy en día por empresas como Google, DELL, digg, NBC, CBS, NETFLIX, mozilla.org, wordpress, drupal, etc.

3.1.- Introducción a jQuery (parte I).

jQuery es un framework JavaScript, que nos va a simplificar muchísimo la programación. Como bien sabes, cuando usamos JavaScript tenemos que preocuparnos de hacer scripts compatibles con varios navegadores y, para conseguirlo, tenemos que programar código compatible.

jQuery nos puede ayudar muchísimo a solucionar todos esos problemas, ya que nos ofrece la infraestructura necesaria para crear aplicaciones complejas en el lado del cliente. Basado en la filosofía de "*escribe menos y produce más*", entre las ayudas facilitadas por este framework están: la creación de interfaces de usuario, uso de efectos dinámicos, AJAX, acceso al DOM, eventos, etc. Además esta librería cuenta con infinidad de plugins, que nos permitirán hacer presentaciones con imágenes, validaciones de formularios, menús dinámicos, drag-and-drop, etc.

Esta librería es gratuita, y dispone de licencia para ser utilizada en cualquier tipo de plataforma, personal o comercial. El fichero tiene un tamaño aproximado de 31 KB, y su carga es realmente rápida. Además, una vez cargada la librería, quedará almacenada en caché del navegador, con lo que el resto de páginas que hagan uso de la librería, no necesitarán cargarla de nuevo desde el servidor.

Página Oficial de descarga de la librería jQuery.
http://jquery.com/

Documentación oficial de la librería jQuery.
http://docs.jquery.com/Main_Page

Para poder programar con jQuery, lo primero que tenemos que hacer es cargar la librería. Para ello, podemos hacerlo de dos formas:

Cargando la librería directamente desde la propia web de jQuery con la siguiente instrucción:

```
<script type="text/javascript" src="HTTP://code.jquery.com/jquery-latest.js"></script>
```

De esta forma, siempre nos estaremos descargando la versión más actualizada de la librería. El único inconveniente, es que necesitamos estar conectados a Internet para que la librería pueda descargarse.

Cargando la librería desde nuestro propio servidor:

```
<script type="text/javascript" src="jquery.js"></script>
```

De esta forma, el fichero de la librería estará almacenado como un fichero más de nuestra aplicación, por lo que no necesitaremos tener conexión a Internet (si trabajamos localmente), para poder usar la librería. Para poder usar este método, necesitaremos descargarnos el fichero de la librería desde la página de jQuery (`jquery.com`). Disponemos de dos versiones de descarga: la *versión de producción* (comprimida para ocupar menos tamaño), y *la versión de desarrollo* (descomprimida). Generalmente descargaremos la versión de producción, ya que es la que menos tamaño ocupa. La versión de desarrollo tiene como única ventaja que nos permitirá leer, con más claridad, el código fuente de la librería (si es que estamos interesados en modificar algo de la misma).

La clave principal para el uso de jQuery radica en el uso de la función `$()`, que es un alias de `jQuery()`. Esta función se podría comparar con el clásico `document.getElementById()`, pero con una

diferencia muy importante, ya que soporta selectores CSS, y puede devolver arrays. Por lo tanto `$()` es una versión mejorada de `document.getElementById()`.

Selectores CSS.

Anexo II - Selectores CSS que deberías conocer

Esta función `$("selector")`, acepta como parámetro una cadena de texto, que será un selector CSS, pero también puede aceptar un segundo parámetro, que será el contexto en el cuál se va a hacer la búsqueda del selector citado. Otro uso de la función, puede ser el de `$(function){..});` equivalente a la instrucción `$(document).ready (function() {...});` que nos permitirá detectar cuando el DOM está completamente cargado.

Verás un ejemplo de cómo usar estas instrucciones en apartado siguiente 3.2.

3.2.- Introducción a jQuery (parte II).

Vamos a ver en este apartado, un ejemplo programado por el método tradicional, y su equivalencia, usando la librería jQuery:

normal.html

```
<!DOCTYPE HTML>
<html>
   <head>
      <meta charset="utf-8">
      <title>Ejemplo sin jQuery</title>
      <style type="text/css">
         .colorido{
            background-color:#99FF33;
         }
      </style>

      <script type="text/javascript" src="funciones.js"></script>
      <script type="text/javascript">
         ////////////////////////////////////////////////////////////////////
         // Cuando el documento esté cargado completamente llamamos a la función iniciar().
         ////////////////////////////////////////////////////////////////////
         crearEvento(window,"load",iniciar);
         ////////////////////////////////////////////////////////

         function iniciar(){
            var tabla=document.getElementById("mitabla"); // Seleccionamos la tabla.
            var filas= tabla.getElementsByTagName("tr");   // Seleccionamos las filas de la
tabla.

            for (var i=0; i<filas.length; i++){
               if (i%2==1){
                  // Es una fila impar
                  // Aplicamos la clase .colorido a esas filas.
                  filas[i].setAttribute('class','colorido');
               }
            }
         }
      </script>
   </head>
   <body>
      <table width="200" border="1" align="center" id="mitabla">
         <tr>
            <td><div align="center"><strong>pais</strong></div></td>
            <td><div align="center"><strong>habitantes</strong></div></td>
            <td><div align="center"><strong>renta</strong></div></td>
         </tr>
         <tr>
            <td>España</td>
            <td>15600000</td>
            <td>25000</td>
         </tr>
         <tr>
            <td>Italia</td>
            <td>45105500</td>
            <td>45000</td>
```

```
         </tr>
         <tr>
            <td>Francia</td>
            <td>58454545</td>
            <td>45645</td>
         </tr>
         <tr>
            <td>Uk</td>
            <td>78799788</td>
            <td>88547</td>
         </tr>
         <tr>
            <td>USA</td>
            <td>98878787</td>
            <td>45124</td>
         </tr>
      </table>
   </body>
</html>
```

jquery.html

```
<!DOCTYPE HTML>
<html>
   <head>
      <meta charset="utf-8">
      <title>Ejemplo con jQuery</title>
      <style type="text/css">
         .colorido{
            background-color:#99FF33;
         }
      </style>

      <script type="text/javascript" src="http://code.jquery.com/jquery-latest.js"></script>
      <script type="text/javascript">
         $(document).ready(function(){
            // Cuando el documento esté preparado se ejecuta esta función.
            // Seleccionamos las filas impares contenidas dentro de mitabla y le aplicamos la
clase colorido.
            $("#mitabla tr:nth-child(even)").addClass("colorido");
         });
      </script>
   </head>
   <body>
      <table width="200" border="1" align="center" id="mitabla">
         <tr>
            <td><div align="center"><strong>pais</strong></div></td>
            <td><div align="center"><strong>habitantes</strong></div></td>
            <td><div align="center"><strong>renta</strong></div></td>
         </tr>
         <tr>
            <td>España</td>
            <td>15600000</td>
            <td>25000</td>
         </tr>
         <tr>
            <td>Italia</td>
            <td>45105500</td>
            <td>45000</td>
         </tr>
         <tr>
            <td>Francia</td>
            <td>58454545</td>
            <td>45645</td>
         </tr>
         <tr>
            <td>Uk</td>
            <td>78799788</td>
            <td>88547</td>
         </tr>
         <tr>
            <td>USA</td>
            <td>98878787</td>
            <td>45124</td>
         </tr>
      </table>
   </body>
</html>
```

funciones.js

```
////////////////////////////////////////////////////////
// Función cross-browser para crear objeto XMLHttpRequest
////////////////////////////////////////////////////////
function objetoXHR(){
   if (window.XMLHttpRequest){
      // El navegador implementa la interfaz XHR de forma nativa
      return new XMLHttpRequest();
   }else if (window.ActiveXObject){
      var versionesIE = new Array('Msxml2.XMLHTTP.5.0', 'Msxml2.XMLHTTP.4.0',
      'Msxml2.XMLHTTP.3.0', 'Msxml2.XMLHTTP', 'Microsoft.XMLHTTP');

      for (var i = 0; i < versionesIE.length; i++){
         try{
            /*
               Se intenta crear el objeto en Internet Explorer comenzando
               en la versión más moderna del objeto hasta la primera versión.
               En el momento que se consiga crear el objeto, saldrá del bucle
               devolviendo el nuevo objeto creado.
            */
            return new ActiveXObject(versionesIE[i]);
         }catch (errorControlado) {}//Capturamos el error,
      }
   }

   /*
      Si llegamos aquí es porque el navegador no posee ninguna forma de crear el objeto.
      Emitimos un mensaje de error usando el objeto Error.

      Más información sobre gestión de errores en:
      http://www.javascriptkit.com/javatutors/trycatch2.shtml
   */

   throw new Error("No se pudo crear el objeto XMLHttpRequest");
}


////////////////////////////////////////////////////////
// Función cross-browser para añadir Eventos
////////////////////////////////////////////////////////
var crearEvento = function(){
   function w3c_crearEvento(elemento, evento, mifuncion){
      elemento.addEventListener(evento, mifuncion, false);
   }

   function ie_crearEvento(elemento, evento, mifuncion){
      var fx = function(){
         mifuncion.call(elemento);
      };

      // Enlazamos el evento con attachEvent. Cuando usamos attachEvent
      // dejamos de tener acceso al objeto this en mifuncion. Para solucionar eso
      // usaremos el método call() del objeto Function, que nos permitirá
      // asignar el puntero this para su uso dentro de la función. El primer
      // parámetro que pongamos en call será la referencia que se usará como
      // objeto this dentro de nuestra función. De esta manera solucionamos el problema
      // de acceder a this usando attachEvent en Internet Explorer.

      elemento.attachEvent('on' + evento, fx);
   }

   if (typeof window.addEventListener !== 'undefined'){
      return w3c_crearEvento;
   }else if (typeof window.attachEvent !== 'undefined'){
      return ie_crearEvento;
   }
}();  // <= Esta es la parte más crítica - tiene que terminar en ()


////////////////////////////////////////////////////////
// Función cross-browser para modificar el contenido
// de un DIV
////////////////////////////////////////////////////////
function textoDIV(nodo, texto){
   //var nodo = document.getElementById(idObjeto);
   while (nodo.firstChild)
      nodo.removeChild(nodo.firstChild); // Eliminamos todos los hijos de ese objeto.
```

```
   // Cuando ya no tenga hijos, agregamos un hijo con el texto que recibe la función.
   nodo.appendChild(document.createTextNode(texto));
}
```

Ejemplo usando el método tradicional con JavaScript:

```
     … Aquí irán las cabeceras y clase .colorido

<script type="text/javascript" src="funciones.js"></script>
<script type="text/javascript">
   ////////////////////////////////////////////////////////////////////
   // Cuando el documento esté cargado completamente llamamos a la función iniciar().
   ////////////////////////////////////////////////////////////////////
   crearEvento(window,"load",iniciar);
   ////////////////////////////////////////////////////////////////////

   function iniciar(){
      var tabla=document.getElementById("mitabla"); // Seleccionamos la tabla.
      var filas= tabla.getElementsByTagName("tr");  // Seleccionamos las filas de la tabla.

      for (var i=0; i<filas.length; i++){
         if (i%2==1){
            // Es una fila impar
            // Aplicamos la clase .colorido a esas filas.
            filas[i].setAttribute('class','colorido');
         }
      }
   }
</script>
</head>
<body>
     .. Aquí irá la tabla …
</body>
</HTML>
```

Ejemplo equivalente al anterior, programado usando la librería jQuery:

```
     … Aquí irán las cabeceras y clase .colorido

<script type="text/javascript" src="HTTP://code.jquery.com/jquery-latest.js"></script>
<script type="text/javascript">
// También podríamos poner $(function) {….});
$(document).ready(function()        // Cuando el documento esté preparado se ejecuta esta
función.
{
     // Seleccionamos las filas impares contenidas dentro de mitabla y le aplicamos la clase
colorido.
     $("#mitabla tr:nth-child(even)").addClass("colorido");
});
</script>
</head>
<body>
     .. Aquí irá la tabla …
</body>
</HTML>
```

Como puedes observar, la reducción de código es considerable. Con 2 instrucciones, hemos conseguido lo mismo, que hicimos en el ejemplo anterior con 7 (sin contar el código de `crearEvento` del fichero `funciones.js`).

3.3.- Función $.ajax() en jQuery.

La principal función para realizar peticiones AJAX en jQuery es **`$.AJAX()`** (importante no olvidar el punto entre `$` y `AJAX()`). Ésta es una función de bajo nivel, lo que quiere decir que disponemos de la posibilidad de configurar, prácticamente todos los parámetros de la petición AJAX, y será, por tanto, equivalente a los métodos clásicos que usamos en la programación tradicional.

La sintaxis de uso es: **`$.AJAX(opciones)`**
En principio, esta instrucción parece muy simple, pero el número de opciones disponibles, es relativamente extenso. Ésta es la estructura básica:

```
$.AJAX({
     url: [URL],
```

```
     type: [GET/POST],
     success: [function callback exito(data)],
     error: [function callback error],
     complete: [function callback error],
     ifModified: [bool comprobar E-Tag],
     data: [mapa datos GET/POST],
     async: [bool que indica sincronía/asincronia]
});
```

Por ejemplo:

```
$.AJAX({
     url: '/ruta/pagina.php',
     type: 'POST',
     async: true,
     data: 'parametro1=valor1&parametro2=valor2',
     success: function (respuesta)
     {
          alert(respuesta);
     },
     error: mostrarError
});
```

Veamos algunas propiedades de la función `$.AJAX()` de jQuery:

Nombre	Tipo	Descripción
`url`	String	La URL a la que se le hace la petición AJAX.
`type`	String	El método HTTP a utilizar para enviar los datos: POST o GET. Si se omite se usa GET por defecto.
`data`	Object	Un objeto en el que se especifican parámetros que se enviarán en la solicitud. Si es de tipo GET, los parámetros irán en la URL. Si es POST, los datos se enviarán en las cabeceras. Es muy útil usar la función serialize(), para construir la cadena de datos.
`dataType`	String	Indica el tipo de datos que se espera que se devuelvan en la respuesta: XML, HTML, JSON, JSONp, script, text (valor por defecto en el caso de omitir dataType).
`success`	Function	Función que será llamada, si la respuesta a la solicitud terminó con éxito.
`error`	Function	Función que será llamada, si la respuesta a la solicitud devolvió algún tipo de error.
`complete`	Function	Función que será llamada, cuando la solicitud fue completada.

Todos los parámetros de uso de la función $.AJAX() de jQuery.
http://api.jquery.com/jQuery.AJAX/

3.4.- El método .load() y las funciones $.post() , $.get() y $.getJSON() en jQuery.

La función `$.AJAX()` es una función muy completa, y resulta bastante pesada de usar. Su uso es recomendable, para casos muy concretos, en los que tengamos que llevar un control exhaustivo de la petición AJAX. Para facilitarnos el trabajo, se crearon 3 funciones adicionales de alto nivel, que permiten realizar peticiones y gestionar las respuestas obtenidas del servidor:

El método .load()

Este método, es la forma más sencilla de obtener datos desde el servidor, ya que de forma predeterminada, los datos obtenidos son cargados en el objeto al cuál le estamos aplicando el método.

Su sintaxis es: `.load( url, [datos], [callback] )`

La función `callback` es opcional, y es ahí donde pondremos la función de retorno, que será llamada una vez terminada la petición. En esa función realizaremos tareas adicionales, ya que la acción por defecto de cargar en un objeto el contenido devuelto en la petición, la realiza el propio método `load()`.

Ejemplos:

```
$("#noticias").load("feeds.HTML");
// carga en el contenedor con id noticias lo que devuelve la página feeds.HTML.

$("#objectID").load("test.php", { 'personas[]': ["Juan", "Susana"] } );
// Pasa un array de datos al servidor con el nombre de dos personas.
```

Cuando se envían datos en este método, se usará el método `POST`. Si no se envían datos en la petición, se usará el método `GET`.

Más información sobre el método `.load()` en jQuery.
http://api.jquery.com/jQuery.post/

La función $.post()

Nos permite realizar peticiones AJAX al servidor, empleando el método `POST`. Su sintaxis es la siguiente:

```
$.post( url, [datos], [callback], [tipo] )
```

Ejemplos:

```
$.post("test.php");

$.post("test.php", { nombre: "Juana", hora: "11am" } );

$.post("test.php", function(resultados) {
     alert("Datos Cargados: " + resultados);
});
```

Más información sobre el método .post() en jQuery.
http://api.jquery.com/jQuery.post/

La función $.get() y $.getJSON()

Hacen prácticamente lo mismo que `POST`, y tienen los mismos parámetros, pero usan el método `GET` para enviar los datos al servidor. Si recibimos los datos en formato JSON, podemos emplear `$.getJSON()` en su lugar.

```
$.get( url, [datos], [callback], [tipo] ) | $.getJSON( url, [datos], [callback], [tipo] )
```

Más información sobre el método .get() en jQuery.
http://api.jquery.com/jQuery.get/

3.5.- Herramientas adicionales en programación AJAX.

Cuando programamos en AJAX, uno de los inconvenientes que nos solemos encontrar, es el de la detección de errores. Estos errores pueden venir provocados por fallos de programación en JavaScript, fallos en la aplicación que se ejecuta en el servidor, etc.

Para poder detectar estos errores, necesitamos herramientas que nos ayuden a encontrarlos. En la programación con JavaScript, los errores los podemos detectar con el propio navegador. Por ejemplo, en el navegador Firefox para abrir la consola de errores, lo podemos hacer desde el menú Herramientas, o bien pulsando las teclas **CTRL + Mayúsc. + J** (en Windows). En la consola, se nos mostrarán todos los errores que se ha encontrado durante la ejecución de la aplicación. En Internet Explorer versión 9, podemos abrir la **Herramienta de Desarrollo**, pulsando la tecla F12. Desde esta herramienta se pueden consultar los errores de JavaScript, activar los diferentes modos de compatibilidad entre versiones de este navegador, deshabilitar CSS, JavaScript, etc.

Para la detección de errores en AJAX, necesitamos herramientas adicionales o complementos. Para Firefox disponemos de un complemento denominadoFirebug. Este complemento nos va a permitir hacer infinidad de cosas: detectar errores de JavaScript, depurar código, analizar todo el DOM del documento en detalle, ver y modificar el código CSS, analizar la velocidad de carga de las páginas, etc. Además, también incorpora en su consola, la posibilidad de ver las peticiones AJAX que se están realizando al servidor: se pueden ver los datos enviados, su formato, los datos recibidos, los errores, etc. Si por ejemplo se produce algún tipo de error en la petición al servidor, en la consola podremos verlo y así poder solucionar ese fallo.

Vídeo sobre el uso de firebugs de firefox para ajax
Vídeo de introducción a node.js
Anexo III - 10 extensiones de Firefox para desarrollo web.

3.6.- Plugins jQuery.

La librería jQuery, incorpora funciones que nos van a ayudar muchísimo, en la programación de nuestras aplicaciones. Además de todo lo que nos aporta la librería, disponemos de plugins o añadidos que aportan funcionalidades avanzadas.

Vamos a encontrar plugins en un montón de categorías: AJAX, animación y efectos, DOM, eventos, formularios, integración, media, navegación, tablas, utilidades, etc.

Anexo IV - Efectos con jQuery.
Documentación oficial de jQuery. **http://docs.jquery.com/Main_Page**

Antes de poder usar cualquier plugin de jQuery, será necesario cargar primero la librería de jQuery, y a continuación la librería del plugin que deseemos, por ejemplo:

```
<script type="text/javascript" src="HTTP://code.jquery.com/jquery-latest.js"></script>

<script type="text/javascript" src="ejemploplugin.js"></script>
```

Todos los plugins contienen documentación, en la que se explica cómo usar el plugin.

Desde la web oficial de `jquery.com`, puedes hojear todos los plugins disponibles para jQuery:
Plugins para jQuery. **http://plugins.jquery.com/**

También es muy común el encontrar páginas, en las que se muestran rankings de los mejores plugins para jQuery. Algunos ejemplos pueden ser:
Los Mejores plugins jQuery del año 2011.
http://www.ajaxline.com/best-jquery-plugins-february-2011
Los 130 mejores plugins de jQuery.
http://pixelcurse.com/jquery/ultimate-roundup-of-best-jquery-plugins
Búsqueda en Google de los mejores plugins de jQuery.
http://www.google.es/#sclient=psy&hl=es&source=hp&q=best+jquery+plugins&aq=f&aqi=g3&aql=&oq=&pbx=1&bav=on.2,or.r_gc.r_pw.&fp=d64bad206e66ecae&biw=1920&bih=888

3.7.- Ejemplos en vídeo, de AJAX con jQuery.

Y para terminar, te vamos a poner unos enlaces a unos vídeos hospedados en Youtube.com

Gestión de una tabla con jQuery
http://www.youtube.com/watch?feature=player_embedded&v=oj-ktOyHzao

Trabajando con hiperenlaces
http://www.youtube.com/watch?feature=player_embedded&v=f5OEAlWx46M
Iconos indicando tipos de hiperenlace
http://www.youtube.com/watch?feature=player_embedded&v=dp4cIWHVWEc
Menú con efectos dinámicos
http://www.youtube.com/watch?feature=player_embedded&v=j5I5Dum4DA8
Efectos Rollover sobre imágenes
http://www.youtube.com/watch?feature=player_embedded&v=hC94Ks8SnFE

Y aquí tienes el código fuente:

fichero zip con los fuentes de todos los ejercicios de los vídeos

Anexo I - Listado de librerías, frameworks y herramientas para AJAX, DHTML y JavaScript

Con esto de las aplicaciones web 2.0, se han desarrollado una gran cantidad de utilidades/herramientas/framework para el desarrollo web con JavaScript, DHTML (HTML dínamico) y AJAX. He aquí el gran listado:

- ✓ Prototype es un *framework* basado en JavaScript que se orienta al desarrollo sencillo y dinámico de aplicaciones web. Es una herramienta que implementa las técnicas AJAX y su potencial es aprovechado al máximo cuando se desarrolla con Ruby On Rails. (fuente)
- ✓ AHAH (Asychronous HTML and HTTP) es un microformato que permite la actualización asíncrona del contenido (X)HTML, y su formateo con CSS, al estilo de lo que hace AJAX. La diferencia con éste es que esto se realiza utilizando (X)HTML y no XML. Pero como (X)HTML puede ser visto como un dialecto de XML, entonces podemos decir que AHAH está incluído en AJAX (por lo que lo de llamarlo AJAX 2.0 es muy sensacionalista y poco estricto). (fuente)
- ✓ dojo es un Framework que contiene APIs y widgets (controles) para facilitar el desarrollo de aplicaciones Web que utilicen tecnología AJAX. Contiene un sistema de empaquetado inteligente, los efectos de UI, drag and drop APIs, widget APIs, abstracción de eventos, almacenamiento de APIs en el cliente, e interacción de APIs con AJAX. Dojo resuelve asuntos de usabilidad comunes como pueden ser la navegación y detección del navegador, soportar cambios de URL en la barra de URLs para luego regresar a ellas(bookmarking), y la habilidad de degradar cuando AJAX/JavaScript no es completamente soportado en el cliente. (fuente)
- ✓ AjaxAC es un marco de trabajo escrito en PHP y que utiliza AJAX para la relación con el servidor. Este *framework* es liberado bajo la licencia de Apache v2.0. (fuente)
- ✓ JSAN - JavaScript Archive Network es una colección de recursos para JavaScript de código abierto.
- ✓ Ajax.NET Professional es uno de las primeras librerías AJAX disponibles para Microsoft ASP.NET y trabaja con .NET 1.1 y 2.0. Puedes encontrar una guía rápida de cómo dar tus primeros pasos en Ajax.NET, en su web oficial.
- ✓ AjaxRequest Library es producto de AjaxToolbox.com, que simplifica y extiende las capacidades del objeto XMLHttpRequest (el corazón de AJAX) y te permite desarrollar tus proyectos, sin tener que preocuparte por los procesos a bajo nivel.
- ✓ ATLAS es un paquete de nuevas tecnologías de desarrollo web que integra un extenso conjunto de librerias "client script" con la rica plataforma de desarrollo del lado del servidor ASP .NET lo que nos va a permitir poder crear aplicaciones que tengan la posibilidad de realizar actualizaciones sobre una página web en el cliente haciendo llamadas directas al servidor Web sin la necesidad de hacer un "Refresco de Página", lo que nos permite poder aprovechar todo el potencial del lado del Servidor haciendo mucho trabajo en el Cliente permitiendo una mejor interacción de nuestros usuarios con los sistemas que desarrollemos. (fuente)
- ✓ Bajax es una pequeña y simple librería JavaScript para usar AJAX en nuestra páginas web. Es independiente del lenguaje de programación. Podemos mostrar contenido dinámico usando comandos simples. (mas info)
- ✓ MochiKit es una biblioteca de clases de propósito general escrita en JavaScript que suministra características de otros lenguajes de programación como Python u Objective-C. (fuente)
- ✓ Code Snippets es un repositorio público de códigos fuente. Permite facilmente crear tu colección personal de códigos/script, categorizarlas con tags y compartirlas con todo el mundo.
- ✓ DHTML API, Drag & Drop for Images and Layers librería JavaScript DHTML la cual agrega funciones de Drag Drop (arrastre/mover) sobre capas (layers) y cualquier imágen. Una librería que no debe faltarnos.
- ✓ DHTMLgoodies.com nos ofrece una gran cantidad de utilidades/scripts de DHTML, JavaScript y Ajax.
- ✓ Dynamic Drive un lugar en la web donde podemos obtener de manera gratuita utilidades/scripts DTHML y JavaScript para agregarlas a nuestros proyectos. Este sitio se actualiza regularmente.

- ✓ DynAPI es una librería, de código abierto, en JavaScript para crear componentes Dinámicos para HTML (DHTML) en una página web.
- ✓ qooxdoo es una librería que ofrece muchas facilidades para crear interfaces javascript avanzados, incluyendo una consola de depuración, manejo de eventos, control del foco… Soporta la mayoría de los navegadores actuales y tiene licencia LGPL. (fuente)
- ✓ Engine for Web Applications es un framework para desarrollo de aplicaciones web del lado del cliente.
- ✓ JavaScript Libraries sitio web donde podemos encontrar gran cantidad de utilidades/scripts en JavaScript y DHTML, tales como: manejo de formularios, retención de variables, cargar/mostrar imágenes, menús, efectos y entre otros como XML/RSS/DOM.
- ✓ Javascript Toolbox es un repositorio de códigos y librerías reutilizables que satisfacer necesidades comunes que enfrentan muchos desarrolladores web. La gran cantidad de estos código es compatible con la mayoría de navegadores. Podemos encontrar códigos fiables, pues son probados y testeados para un correcto funcionamiento. Excelente iniciativa realmente!.
- ✓ Taconite es framework que simplifica la creación de aplicaciones web Ajax. Automatiza las tediosas tareas relacionadas con Ajax, tales como la creación y gestión del objeto XMLHttpRequest y la creación de contenido dinámico. Taconite se puede utilizar con todos los navegadores web actuales (Firefox, Safari, Internet Explorer, Opera y Konqueror, por citar algunos) y puede utilizarse con tecnologías del lado del servidor como Java EE, .Net, PHP ó cualquier lenguaje que retorne como respuesta XHTML.
- ✓ jQuery es un nuevo tipo de librerias de Javascript que permite simplificar la manera de interactuar con los documentos HTML, permitiendo manejar eventos,desarrollar animaciones, y agregar interacción con la tecnología AJAX a nuestras páginas web. jQuery esta diseñado para cambiar la forma de escribir código JavaScript. (fuente)
- ✓ JSL: JavaScript Standard Library es un único y pequeño archivo (7.7 KB) con funciones y métodos estándar de JavaScript. Compatible con cualquier navegador que soporte al menos JavaScript 1.2.
- ✓ DHTML Kitchen es un sitio web donde podemos encontrar muchos códigos/script e información sobre DHTML.
- ✓ liberty es una librería básica (simple) para desarrollo web con JavaScript. (fuente)
- ✓ moo.fx es un librería Javascript liviana y pequeña (3KB) con la cual podemos conseguir unos efectos muy interesantes. Trabaja con los frameworks Prototype y Mootools. Simple y fácil de usar. Podemos controlar ó modificar las propiedades CSS y los elementos HTML.
- ✓ overLIB es una librería JavaScript que nos permite mostrar una pequeña caja de información (popup) sobre los enlaces ó link de nuestras páginas web. Brindan asó información a nuestros usuarios sobre a donde nos llevan los links.
- ✓ TurboWidgets son controles JavaScript del lado del cliente para proporcionan un agradable y manejable interfaz de usuario para aplicaciones web estilo AJAX. Construido con Dojo Toolkit, TurboWidgets están diseñados para un uso fácil.
- ✓ overlibmws DHTML Popup Library es una librería DHTML, cuenta con documentación y muchos ejemplos.
- ✓ PlotKit - Javascript Chart Plotting librería en JavaScript para la creación de gráficos. Es soportado por el elemento HTML Canvas, SVG y soporte nativo del navegador. Plokit cuenta con documentación y ejemplos para hacer usarlo en nuestros proyectos sin inconvenientes.
- ✓ qForms JavaScript API es uno de los más completas API JavaScript para la fácil creación y manipulación de formularios en nuestro proyectos web.
- ✓ Zapatec AJAX Suite te brinda una cantidad de herramientas para interfaces de usuarios en tus aplicaciones web, como por ejemplo: calendarios, menús, explorador árbol, formularios, grid, slider, tabs, drag-drgop, efectos y más.
- ✓ Rico es una librería de efectos Ajax disponible en OpenRico que permite simplificar el desarrollo de aplicaciones que utilicen esta tecnología. Mediante Rico es muy sencillo definir la operación básica de Ajax: enviar una solicitud al servidor para que devuelva información. Dispone también de algunos efectos gráficos, tablas actualizables y secciones de drag & drop. (fuente)

- ✓ Sajax es una herramienta de código abierto diseñada para ayudar a los sitios web que usan AJAX framework (también conocido como XMLHttpRequest). Permite al programador llamar a funciones PHP, Perl o Python desde su página web por medio de JavaScript sin necesidad de forzar una actualización de la página en el navegador. (fuente)
- ✓ sardalya herramienta API la creación de páginas DHTML, diseñada para trabajar en todos los navegadores que soportan DOM.
- ✓ script.aculo.us es una librería JavaScript que permite el uso de controles AJAX, drag & drop, y otros efectos visuales en una página web. Se distribuye mediante descargas en varios formatos de archivo, y también está incluido en Ruby on Rails y otros frameworks de desarrollo web.
- ✓ Spry Framework for Ajax es una librería JavaScript de Adobe que facilita el uso de funciones con AJAX. Se encarga de manejar la complejidad interna del AJAX y permite al desarrollador crear facilmente aplicaciones web 2.0.
- ✓ Tacos librería que proporciona componentes AJAX para Tapestry (framework para el desarrollo aplicaciones web en Java). Su funcionalidad está basada en el framework Dojo.
- ✓ TwinHelix nos ofrece proyectos libres DHTML y JavaScript, aunque también XHTML, CSS y CGI.
- ✓ Yahoo! User Interface Library es un paquete de utilidades y controles, escritos en JavaScript, que facilitan la construcción de aplicaciones interactivas (RIA). [Tales como] Drag and drops, animaciones, aplicaciones con Ajax, DOM, etc. Todas muy completas y fáciles de poner en práctica (con pocas líneas de código). La finalidad de esta librería (y de ahí el nombre) es facilitar el desarrollo de aplicaciones ricas del lado del cliente (usuario), logrando elementos visuales e interactivos que incluyen CSS. (fuente)
- ✓ Zebda es una librería en JavaScript para diversos propositos. Se basa en Prototype 1.4.0.
- ✓ Zephyr es un framework para crear aplicaciones AJAX con PHP5. Puedes desarrollar fácilmente aplicaciones empresariales utilizando este robusto framework. Es muy fácil de aprender y muy sencillo de implementar.
- ✓ ZK es un framework Ajax de código abierto que dispone de herramientas ó controles para crear interfaces de usuarios similares a las de escritorio.
- ✓ ext es un framework del lado del cliente para el desarrollo de aplicaciones web. Tiene un sistema dual de licencia: Comercial y Opern Source. Este framework puede correr en cualquier plataforma que pueda procesar POST y devolver datos estructurados (PHP, Java, .NET y algunas otras). (fuente)
- ✓ mootools es un framework JavaScript compacto y modular, orientado a objeto para la creación de aplicaciones web compatible con cualquier navegador.
- ✓ ¿Cónoces de alguna otra librería ó framework para JavaScript, DHTML y AJAX?
  Basado en AJAX, DHTML and JavaScript Libraries.

Anexo II - Selectores CSS que deberíamos conocer

Con la masiva utilización de CSS como sistema de maquetación de páginas web están apareciendo gran cantidad de utilidades y "trucos" para optimizar nuestras hojas de estilos. Esto nos ofrece un mayor control sobre los elementos de nuestro HTML sin necesidad de sobrecargar el HTML con `clases` y `ID's` que realmente no necesitamos.

Para conseguir parte de estas mejoras, deberemos usar los llamados **selectores**, que en resumen son una forma de permitirnos elegir un elemento (o varios) entre todos los que tenemos en nuestro HTML. Similar al funcionamiento de las expresiones regulares para el texto, los selectores nos permiten usar caracteres especiales para referirnos a un elemento o un rango de los mismos.

Selector	Descripción
*	Selector universal, son todos los elementos del CSS
E	`E` representa cualquier elemento del tipo `E` (`span`, `p`, …)
E F	Todos los elementos `F` que sean descendentes de `E`
E > F	Todos los elementos `F` que sean hijos de `E`
E:first-child	De esta forma podemos seleccionar el primer elemento de tipo `E`
E:link , E:visited	Selecciona los elementos `E` que sean un enlace y no hayan sido visitados (`:link`) y los si visitados (`:visited`)
E:active , E:hover , E:focus	Selecciona los elementos de tipo `E` , en sus correspondientes acciones.
E:lang(c)	Cogemos los elementos del tipo `E` que estén en el idioma (humano) especificado en `(c)`.
E + F	Se trata de cualquier elemento `F` inmediatamente después del elemento del tipo `E`
E[foo]	Elementos del tipo `E` con el atributo `foo`
E[foo="ejemplo"]	Elementos del tipo `E` con el atributo `foo` igual a "`ejemplo`"
E[foo~="ejemplo"]	Elementos del tipo `E` con el atributo `foo` contenga "`ejemplo`". Se pueden añadir varias palabras separadas por espacios. (~ =ALT + 0126)
E[lang|="es"]	Similar al anterior, pero se referirá a todos los elemento `E` tal que su atributo `lang` comienze por "`es`". Por ejemplo: "`es_ES`", "`es_CA`",…
E[foo$="ejemplo"]	Elementos del tipo `E` en el que el atributo `foo` termine con "`ejemplo`".
DIV.ejemplo	Todos los elementos `DIV` que sean de la clase `ejemplo`
E#miID	El elemento `E` en el que su `ID` sea igual `miID`

Ampliando los selectores

Además de estos no pocos selectores podemos, aún más, conseguir filtrar la búsqueda de elementos, para ello usaremos pseudo-elementos.

`:first-line`

Se refiere a la primera línea del elemento, normalmente usado para elementos de texto.

```
p {font-size: 12pt}
p:first-line {color: #0000FF; font-variant: small-caps}

<p>Some text that ends up on two or more lines</p>
```

Propiedades:

- ✓ font properties
- ✓ color properties
- ✓ background properties
- ✓ word-spacing
- ✓ letter-spacing
- ✓ text-decoration

- ✓ vertical-align
- ✓ text-transform
- ✓ line-height
- ✓ clear

:first-letter

La primera letra del elemento, también suele usarse para elementos de texto.

```
p {font-size: 12pt}
p:first-letter {font-size: 200%; float: left}

<p>The first words of an article.</p>
```

Propiedades:

- ✓ font properties
- ✓ color properties
- ✓ background properties
- ✓ margin properties
- ✓ padding properties
- ✓ border properties
- ✓ text-decoration
- ✓ vertical-align (only if 'float' is 'none')
- ✓ text-transform
- ✓ line-height
- ✓ float
- ✓ clear

:before

Elemento usado para insertar algún contenido delante de un elemento.

```
h1:before { content: url(beep.wav) }
```

:after

Elemento usado para insertar algún contenido al final del elemento.

```
h1:after { content: url(beep.wav) }
```

pseudo-elementos y CSS clases.

Nos permite encadenar varios selectores para conseguir focalizar en un elemento un pseudo-elemento concreto.

```
p.article:first-letter {color: #FF0000}
<p class="article">A paragraph in an article</p>
```

Multiples pseudo-elementos

Además nos permite utilizar varios pseudo-elementos sobre un mismo elemento.

```
p {font-size: 12pt}
p:first-letter {color: #FF0000; font-size: 200%}
p:first-line {color: #0000FF}

<p>The first words of an article</p>
```

Compatibilidad

Pseudo-elemento	IE	F	N	W3C
:first-letter	5	1	8	1
:first-line	5	1	8	1
:before		1.5	8	2
:after		1.5	8	2

Anexo III - 10 extensiones de firefox para el desarrollo web

Sin lugar a dudas, una de las mejores cosas de **Firefox** es la posibilidad de agregar **extensiones**. Aquí va un listado de 10 extensiones bastante útiles para cuando estás trabajando en tu blog u otro sitio web:

1. **Web Developer**, probablemente el más clásico de todos. Una barra con un montón de funcionalidades, desde proporcionar información sobre elementos, herramientas de validación, reglas y guías, ajustar el tamaño de la ventana, delinear elementos (¡muy práctico!), etc.
2. **Firebug**, el próximo gran clásico. Si bien es cierto que repite algunas de las funcionalidades del anterior, también lo es que agrega muchas más, en especial sus herramientas para trabajar con JavaScript, **DOM**, actividad de la red, etc.
3. **YSlow**, un complemento para el complemento anterior. YSlow analiza una página de acuerdo a las directrices utilizadas por **Yahoo** para sitios de alto rendimiento y entrega un detallado reporte junto a algunas herramientas. Ideal para detectar y prevenir "cuellos de botella" en la carga de tu web.
4. **Screengrab!**, simplemente, la mejor herramienta que he probado para crear y guardar *screenshots* de una ventana, la porción visible o toda la página.
5. **IE Tab**, integra el motor de renderizado de **MSIE** en Firefox, con sólo un click es posible cambiar entre **Gecko** e Internet Explorer.
6. **Dust-Me Selectors**. La versión corta: una extensión para encontrar selectores CSS que no se utilizan. La versión larga: extrae todos los selectores de las hojas de estilo de la página que se está viendo y luego analiza la página para comparar qué selectores no se están utilizando.
7. **ShowIP**, como su nombre lo indica, muestra la dirección **IP** y permite consultar algunos servicios por IP o nombre del dominio, como **whois**, **Netcraft**, **traceroute** o localización geográfica.
8. **Make Link**, agrega una entrada al menú desplegado con el botón secundario del mouse para crear fácilmente enlaces en formato **HTML**, BBCode (para foros) o como simple texto. Además es posible crear nuestras propias combinaciones en base a algunas variables, con lo que las posibilidades son infinitas.
9. **FireFTP**, todo un cliente de **FTP** dentro de Firefox. Especialmente indicado para instalaciones **portables**.
10. **Mouse Gestures**, la única extensión de "uso general" que se me ha hecho verdaderamente indispensable. Permite ejecutar comandos comunes con simples gestos de ratón; recomiendo especialmente habilitar los gestos *rockers* con la rueda.

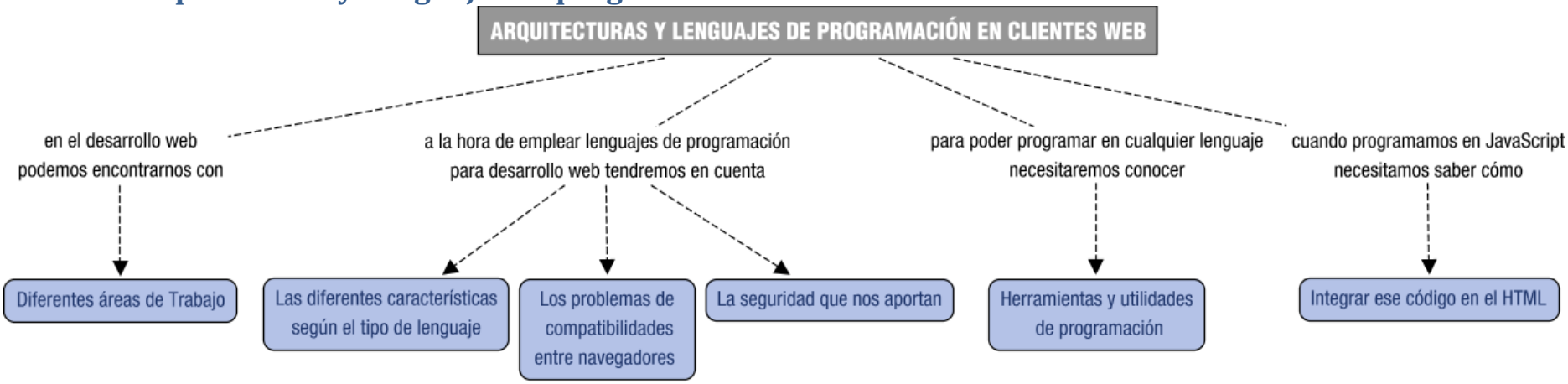
Anxo IV - Efectos con jQuery

Category: Effects

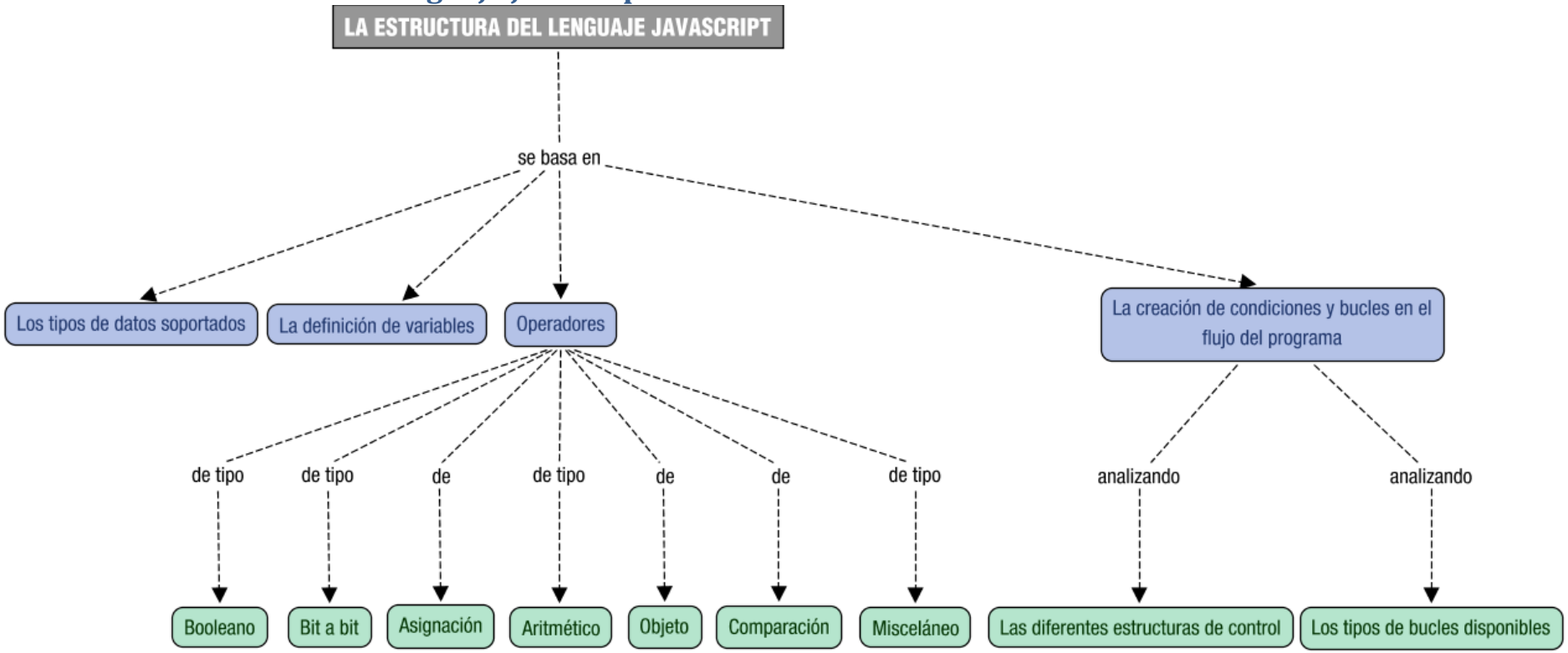
The jQuery library provides several techniques for adding animation to a web page. These include simple, standard animations that are frequently used, and the ability to craft sophisticated custom effects.

.animate()
Perform a custom animation of a set of CSS properties.
.clearQueue()
Remove from the queue all items that have not yet been run.
.delay()
Set a timer to delay execution of subsequent items in the queue.
.dequeue()
Execute the next function on the queue for the matched elements.
.fadeIn()
Display the matched elements by fading them to opaque.
.fadeOut()
Hide the matched elements by fading them to transparent.
.fadeTo()
Adjust the opacity of the matched elements.
.fadeToggle()
Display or hide the matched elements by animating their opacity.
.finish()
Stop the currently-running animation, remove all queued animations, and complete all animations for the matched elements.
.hide()
Hide the matched elements.
jQuery.fx.interval
The rate (in milliseconds) at which animations fire.
jQuery.fx.off
Globally disable all animations.
.queue()
Show or manipulate the queue of functions to be executed on the matched elements.
.show()
Display the matched elements.
.slideDown()
Display the matched elements with a sliding motion.
.slideToggle()
Display or hide the matched elements with a sliding motion.
.slideUp()
Hide the matched elements with a sliding motion.
.stop()
Stop the currently-running animation on the matched elements.
.toggle()
Display or hide the matched elements.

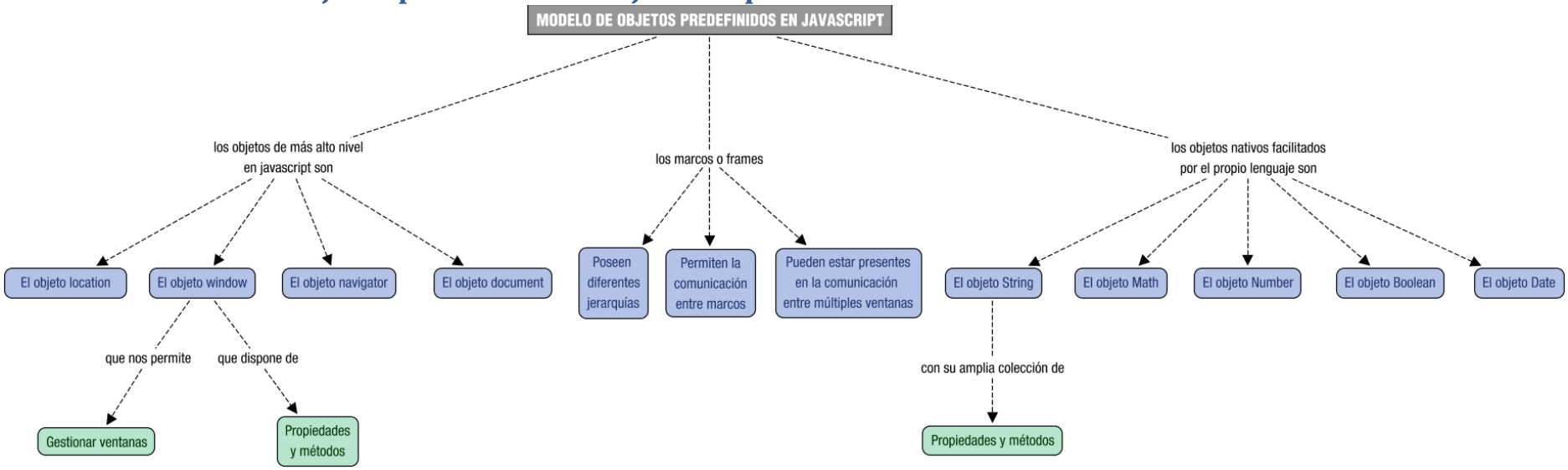
Tema 1 - Arquitecturas y Lenguajes de programación en clientes web



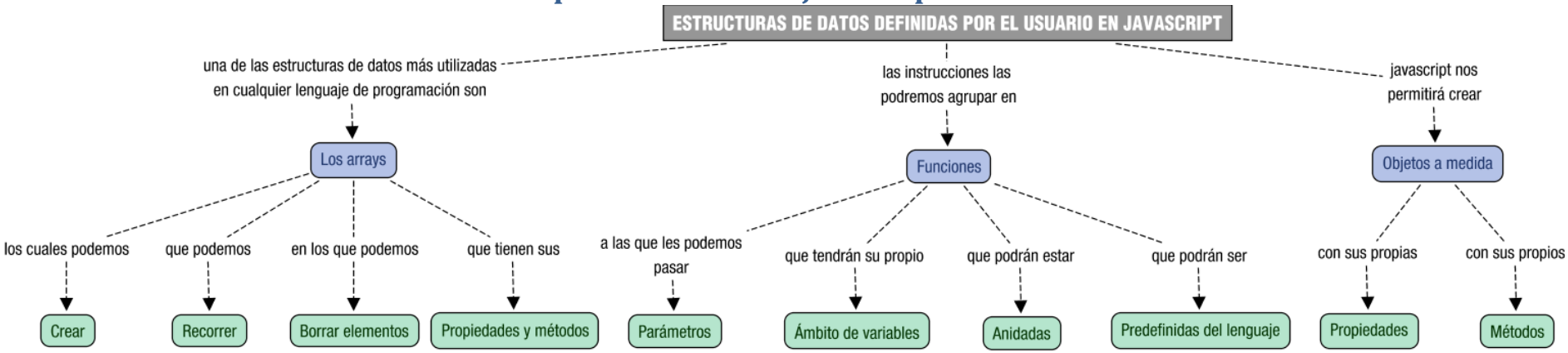
Tema 2 - La estructura del lenguaje javascript



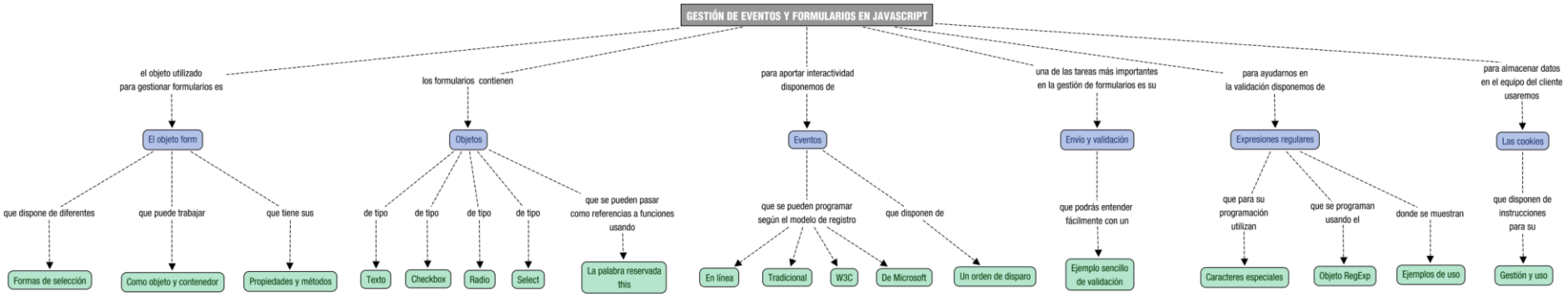
Tema 3 - Modelo de objetos predefinidos en javascript



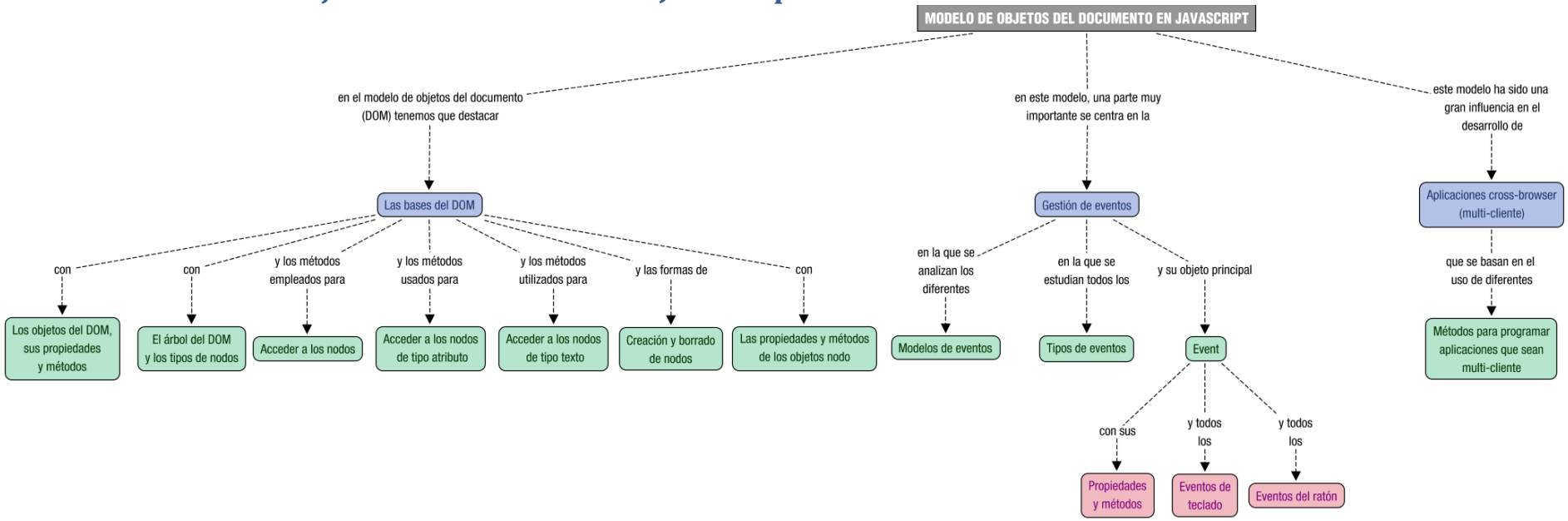
Tema 4 - Estructura de datos definidas por el usuario en javascript



Tema 5 - Gestión de eventos y formularios en JavaScript



Tema 6 - Modelo de objetos del documento en JavaScript



Tema 7 - Programación AJAX en JavaScript