Desarrollo web en Entorno Cliente

DESARROLLO DE APLICACIONES WEB

TEMA 1 – ARQUITECTURAS Y LENGUAJES DE PROGRAMACIÓN EN CLIENTES WEB

Eva Rabasco Bravo
Curso: 2018/19

ÍNDICE

1.- Desarrollo 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. Como resultado se crearon unos protocolos 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.

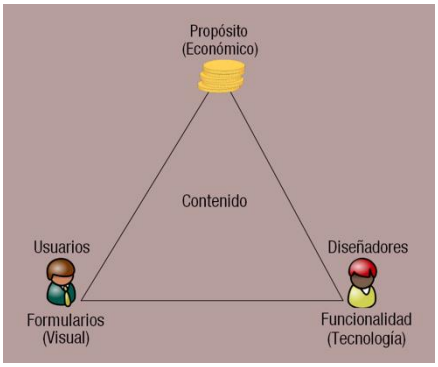
Muchas de las discusiones sobre Diseño Web o Desarrollo Web son confusas, ya que la expresión varía considerablemente. Mientras que la mayoría de la gente tiene algún tipo de noción sobre **lo que significa Diseño Web**, solamente unos pocos son capaces de definirlo con exactitud, y tú vas a estar dentro 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 de usuario , la usabilidad y una variedad de múltiples recursos. Por lo tanto el Desarrollo Web es realmente un campo multidisciplinar.

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 inferior. 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 programas CGI escritos en PERL hasta aplicaciones multihilo 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 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.

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, Ruby On Rails 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.
  ○ Pepper para Chrome.
  ○ 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).
	  ○ JavaScript (Librería Node.js)
	  ○ Ruby (Ruby on Rails)
	...

Hemos escogido JavaScript por que es el lenguaje de script más utilizado en la programación en el lado del cliente, y está soportado mayoritariamente por todas las plataformas. Por lo tanto a partir de ahora todas las referencias que hagamos estarán enfocadas hacia JavaScript.

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.

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 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: Adobe Acrobat, Dreamweaver, OpenOffice.org, Google Docs, Plasmoid (Widgets del Escritorio Linux KDE) etc.

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.

Para saber más Listado de los navegadores web que hay en el mercado con soporte de JavaScript: Comparación de navegadores web.

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: Validador W3C.

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.

2.3.- Seguridad.

JavaScript proporciona un gran potencial para diseñadores maliciosos que quieran distribuir sus scripts a través de la web. Para evitar ésto 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 cuál 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 si 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 cuál 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. Éstas 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.

Cómo ya sabéis, no todos los navegadores procesan el HTML de la misma forma. Tampoco todos ejecutan el código JavaScript al 100% igual. Este tipo de problemas se produce por el motor que implementa cada navegador.

Los motores HTML más populares son:

- Webkit es la base de navegadores como Safari y Chrome.
- Gecko es la base de Firefox.
- Trident la de Internet Explorer
- EdgeHTML se utiliza para el nuevo navegador Microsoft Edge.

Los motores JavaScript son distintos al motor HTML. Por ejemplo, Safari no utiliza el mismo motor Javascript que Chrome.

- JavaScriptCore es utilizado por Safari.
- V8 es utilizado por Chrome.
- SpiderMonkey es utilizado por Firefox.
- Chakra es utilizado por Internet Explorer y por el navegador Edge. Lógicamente utilizan versiones distintas.

Podríamos citar varios ejemplos de motores de JavaScript fuera del ámbito de los navegadores:

- 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 introdujo 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.
- Plasmoid permite programar en JavaScript los elementos gráficos conocidos como Widgets en KDE para Linux. Gnome Shell en Linux también permite programar con JavaScript.
- Algunos motores de programación de videojuegos utilizan JavaScript para programar la parte de la lógica del videojuego. Por ejemplo Unreal Engine y Unity 3D.

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).

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 vamos a dar algunas pistas para 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: Brackets, Sublime Text, Notepad++, Light Table, Aptana Studio, Bluefish, Eclipse, Visual Studio Code, NetBeans, etc.
- Para Macintosh tienes: Aptana Studio, Light Table, Light Table, Bluefish, Eclipse, KompoZer, Nvu, Visual Studio Code , etc.
- Para Linux tienes: Brackets, Sublime Text, Light Table, Geany, KompoZer, Amaya, Quanta Plus, Bluefish, codetch, Visual Studio Code 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.

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, Edge como novedad, etc.
- Para Macintosh tienes: Mozilla Firefox, Safari, Google Chrome, Internet Explorer, etc.
- Para Linux tienes: Mozilla Firefox, Chrome, Chromium, 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 de JavaScript 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 página web que realices. El Validador será tu amigo y te permitirá realizar unas páginas más consistentes.

La dirección del **Validador W3C** es: Validador W3C

Ofrece 3 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.

Más información sobre:
HTML 4.01
HTML5
XHTML 1.0
Especificación sobre:
CSS 2.1
CSS 2.2
Validador del lenguaje CSS
Resumen textual alternativo

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

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 <span lang="en">JavaScript</span> 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 de JavaScript 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. Ésto 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.

4.1.- Integración de código JavaScript con HTML (II).

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 <span lang="en">JavaScript</span> 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 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.

Hay que tener en cuenta que XHTML ya no se sigue desarrollando prefiriéndose la versión 5 de HTML ya que además de ser la favorita por W3C añade muchas nuevas características como son:

- Geolocalización.
- Web Storage: se pueden almacenar datos en el cliente (por ejemplo en una base de datos sqlite).
- Web Workers: hebras para poder poder realizar múltiples tareas concurrentemente.

Además ya no es necesario especificar el atributo type. Tampoco el obsoleto lang. El primer código de esta página quedaría como sigue:

```
<script>
//  código de <span lang="en">JavaScript</span> a continuación
</script>
```

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" del script que has programado para facilitar su carga rápida.

Ofuscador de JavaScript visita: Herramienta para ofuscar JavaScript.

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.

Explicación de licencias Creative Commons.

5.- Fundamentos 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 documento ECMA-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 cuál 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 fundación Mozilla.

Para saber más
Más información sobre Brendan Eich.

5.1.- 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 ésto 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
}
```

Más información y ejemplos sobre comentarios en JavaScript.

5.2.- 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. Aunque se pueden declarar variables sin utilizar var no es una buena práctica. Puedes provocar muchísimos problemas ya que al declararlas así se convierten en variables globales automáticamente incluso dentro de funciones.

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. Bueno, en realidad si que se puede utilizar $ como comienzo de una variable, pero no es buena práctica para nuestros códigos.

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.

5.3.- 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.

5.3.1.- 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.

5.4.- 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 cuál 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

Más información y ejemplos de Operadores.

5.4.1.- Operadores de comparación.

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.

```
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.

```
"123" === 123       // false
```

Más información y ejemplos de Operadores de Comparación.
Diferencias entre igualdad estricta e igualdad.

5.4.2.- Operadores aritméticos.

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.

```
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, x = 2 e = 8.<br />z = z + y;     // z = 14, e y = 8
```

5.4.3.- Operadores de asignación.

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

Más información y ejemplos de Operadores.

5.4.4.- 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).

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
```

5.4.5.- 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.

Ejemplos de operaciones a nivel de bits: Operaciones a nivel de bits.

5.4.6.- 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. <br />// 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:

```
<button id="nombre" onclick="document.write(this.value);" value="Adios"><br> Hola y ...</button><br><br><br />// <code>Tenemos un botón que al pulsarlo escribe adiós. Escribe this.value, que en el objeto actual que es botón // con id nombre. Es decir el value del objeto actual.
```

5.4.7.- 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);
}
```

5.5.- 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.

En el siguiente **test interactivo** podrás comprobar tus conocimientos sobre operadores, condiciones y bucles en JavaScript: http://javascript.simugalicia.com/dwec02r09/

5.5.1.- 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.");
}
```

Construcción switch

Cuando el código se vuelve confuso al utilizar **if ... else** podemos utilizar una estructura condicional es el momento de utilizar switch. Podemos comprobar distintos posibles valores de una variable y actuar en consecuencia.

Sintaxis:

```
switch (variable) {
valor1:  // Instrucciones que se van a realizar
       break; // Rompemos para no ejecutar el resto
valor2: // Instrucciones que se van a realizar
      break;
 .....   // Más comprobaciones
valorn: // Instrucciones a realizar
     break;
default:   // Que se hacen en le resto de los casos
        // No hay break ya que es la última.
}
```

break rompe tanto bucles como switch y deja de ejecutar ese bloque. Utilizar break no es una buena práctica, salvo en switch.

```
accion=prompt("¿Qué desea realizar?","Nada"); // Esta acción va a pedir una variable al usuario a través de una
                                              // ventana similar a alert, pero que permite introducir datos.

switch (accion) {
   case "Saltar": document.write("Acabas de Saltar"); break;
   case "Gritar": document.write("Es imposible que puedas gritar más fuerte."); break;
   case "Trotar": document.write("Pareces un caballo de carreras."); break;
   case "Picar": document.write("Le coges a tu compañero una bolsa de patatas"); break;
   default: document.write("Elige algo que puedas hacer la próxima vez");
}
// Se puede comprobar, no es necesario que sean números enteros.<br />
```

Más información y ejemplos de if..else. Más información y ejemplos de switch

5.5.2.- 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);
```

5.5.3.- 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>

<html>
  <head>
<code>      <title>Ejemplo Básico de JavaScript</title>
     <meta charset="UTF-8">
     <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>
<body>
<h2>Código fuente de un ejemplo básico con JavaScript
</h2>
<script type="text/javascript"><br />   // el type="text/javascript" es opcional a partir de html5
  // Declaramos las variables.
   var nombre, apellidos, edad, hermanos;

   nombre="Rafa";
   apellidos="Martínez 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>
```

Más información y ejemplos sobre JavaScript.

6.- Depuración y errores comunes en JavaScript.

JavaScript es un lenguaje de programación bastante sencillo, con una sintaxis sencilla y familiar tanto a los programadores de C, Java e incluso C#. Sin embargo es bastante traicionero. La mayoría de las veces no te imaginas donde se ha producido el error.

Por ejemplo si una variable no existe y la asignas a un valor lo que vas a producir es un error complicado de depurar. Las buenas prácticas como declarar las variables al principio de los bloques te van a ahorrar un montón de problemas.

6.1.- Depuradores e instalación.

Casi todos los navegadores modernos llevan un depurador más o menos limitado. Sin embargo debemos destacar:

- Firebug. Funciona en distintos navegadores con el Firefox.
- Depurador de Chrome. Interesante, e incluso llegó a sustituir durante bastante tiempo al depurador Firebug.
- Depurador de Firefox. Hasta hace poco era casi obligatorio instalar Firebug en Firefox. Ahora es recomendable pero no obligatorio.

Para unificar la depuración se ha elegido utilizar Firebug. En combinación con Firefox es el que más información sobre el error muestra, haciéndolo el más versátil y recomendado para aprender.

Ahora debes elegir entre Firefox y Chrome:

- Si deseas utilizar Firefox puedes obtener el plugin o extensión de firebug en getfirebug
- Si deseas utilizar Chrome puedes obtener el plugin de firebug lite en getfirebuglite

Una vez instalado puedes pulsar el insecto que representa firebug o pulsar F12 o buscar en los menús interiores un menú de desarrollador.

Además de estas extensiones se puede depurar en otros sistemas, incluido el en línea.

- JSBIN. Es un depurador online bastante bueno, además de ser colaborativo.
- Vorlon.JS. Un nuevo depurador que merece la pena estudiar.
- Herramientas de desarrollo de Internet Explorer desde aquí. A veces no queda más remedio que utilizar los navegadores de Microsoft porque el fallo solo se produce en el suyo.

6.2.- Depuración.

La mejor forma de depurar es aprender a realizarlo. Para ello vamos a seguir unos ejemplos.

Si ejecutamos el script que aparece al inicio del punto 6 se obtendría un mensaje similar a este: "**ReferenceError: nombre is not defined**". Por desgracia los depuradores muestran mensajes en inglés. Sin embargo son fácilmente traducibles. Por ejemplo, aquí pone ReferenceError que significa que algo no se está llamando o se ha hecho una referencia correctamente. nombre es el nombre de la variable y por último no está definido.

Una forma interesante de saber el valor de una variable, sin mostrarlo en pantalla, es usar el método console.log.

```
var nombre="Ada";

console.log("El nombre es" + nombre);<br />// El resultado es:<br />// El nombre es Ada
```

En la consola aparecerá el mensaje y de esa forma evitarás utilizar alert() para mostrar mensajes.

Existe una instrucción del lenguaje llamada debugger que permite que el programa se pare en el punto indicado.

7.- Enlaces de refuerzo y ampliación

- Tutorial de HTML5: http://www.w3schools.com/html/html5_intro.asp
- Integración de Javascript en HTML5: http://www.w3schools.com/js/js_whereto.asp
- Sintaxis general: http://www.w3schools.com/js/js_syntax.asp
- Cometarios: http://www.w3schools.com/js/js_comments.asp
- Variables: http://www.w3schools.com/js/js_variables.asp
- Tipos de datos: http://www.w3schools.com/js/js_datatypes.asp
- Operadores:
  - Operadores en general.http://www.w3schools.com/js/js_operators.asp
  - Operadores artiméticos.http://www.w3schools.com/js/js_arithmetic.asp
  - Operadores de Asignación.http://www.w3schools.com/js/js_assignment.asp
  - Operadores booleanos.http://www.w3schools.com/js/js_booleans.asp
  - Operadores de comparación.http://www.w3schools.com/js/js_comparisons.asp
- Condicional "if": http://www.w3schools.com/js/js_if_else.asp
- Condicional "switch": http://www.w3schools.com/js/js_switch.asp
- Bucle "for": http://www.w3schools.com/js/js_loop_for.asp
- Bucles "while" y "do-while": http://www.w3schools.com/js/js_loop_while.asp
- Break y continue no son recomendables para una buena programación, pero para entenderlos mire aquí: http://www.w3schools.com/js/js_break.asp
- Depuración con las herrmientas de desarrollo de google chrome: https://developers.google.com/web/tools/chrome-devtools/javascript/
- Framework de realización de pruebas unitarias, para facilitar la ejecución y comprobación de código JavaScript: https://qunitjs.com/

Para saber más fuera de este módulo.

Existe una versión de JavaScript que permite utilizar tipos de datos. Este superconjunto se llama TypeScript.

- Página oficial de Typescript: https://www.typescriptlang.org/
- Tipos en español de Typescript: https://www.uno-de-piera.com/tipos-en-typescript/

JavaScript no está limitado al uso de aplicaciones web. Se puede utilizar para realizar aplicaciones en el lado del servidor o incluso android.

- Node. El JavaScript en el lado del Servidor. https://nodejs.org/es/
  - Tutorial de Node. http://www.nodebeginner.org/index-es.html

- APPs para Android en JavaScript con "Protocoder": http://protocoder.org/
  - Programación del ejemplo "HOLA MUNDO" en Protocoder: http://diwo.bq.com/introduccion-a-protocoder-y-hola-mundo/
  - Creación de una interfaz de usuario en Protocoder: http://diwo.bq.com/creando-la-interfaz-de-usuario-variables-y-funciones/
