

MÓDULO

DISEÑO DE PÁGINAS WEB

MARYEM A. RUÍZ NÚÑEZ
JOHN JAIRO MONSALVE RESTREPO



POLITÉCNICO COLOMBIANO
JAIME ISAZA CADAVID



Alcaldía de Medellín

PLAN DE TRABAJO

INTRODUCCIÓN

El programa de Diseño de páginas Web a desarrollarse, comprende tres unidades de trabajo teórico prácticas. Con ellas, se pretende que el alumno posea una visión clara, sencilla, general y completa sobre el diseño y construcción de páginas Web.

El presente módulo de diseño de páginas Web, plantea los siguientes interrogantes generales, para ser resueltos en el desarrollo de esta asignatura.

¿Qué es el WWW?

¿Cómo se estructura un desarrollo Web?

¿Qué es HTML?

¿Cómo se da formato al texto o a un sitio Web?

¿Qué son Marcos y Formularios?

¿Qué es una página de estilo en cascada - CSS?

¿Qué es JavaScript?

¿Cómo se utilizan las CSS y Javascript en un desarrollo Web?

Los anteriores interrogantes se resolverán en el desarrollo de este módulo de la manera más sencilla posible y didácticamente agrupados en unidades. En cada unidad se colocarán actividades a desarrollar en forma práctica, permitiendo que el estudiante interiorice el conocimiento por sus propias vivencias, de igual manera, que surjan nuevos interrogantes y tenga capacidad para resolverlos con propiedad.

Es necesario dejar claro, que el modulo no pretende ser un nuevo manual o texto guía, para el diseño y desarrollo de páginas Web, ya que de estos existen muchos en Internet, por ello tomamos apuntes de varios de estos buenos sitios (manuales) y lo único que se hizo fue darles un orden y adaptación a las necesidades del presente curso. Los sitios más utilizados para elaborar este modulo fueron: <http://www.webestilo.com/html/> - <http://www.desarrolloweb.com/manuales/21> - www.monografias.com (Tom Nissim Zambrano Guillen), a quienes agradecemos por sus buenos trabajos.

OBJETIVO GENERAL

Introducir a los estudiantes, en el diseño y desarrollo de páginas Web.

OBJETIVOS ESPECÍFICOS

- Conocer el lenguaje de desarrollo de páginas web: HTML
- Conocer métodos para obtener diseños uniformes
- Conocer los conceptos de las CSS (Páginas de estilo en Cascada)
- Conocer el Lenguaje de desarrollo Javascript
- Aplicar los conocimientos adquiridos a la resolución de problemas en diversas disciplinas
- Desarrollar páginas web que puedan ser empleadas en el ámbito escolar o comercial

INTEGRACIÓN DEL MÓDULO POR UNIDADES

- **Unidad 1: Concepto Básicos**

Conceptos básicos
Organización de una WEB
Estructura de un desarrollo WEB
El lenguaje HTML
Editores y convertidores
Estructura básica de un documento HTML
Dando forma al texto del documento HTML
Enlaces
Imágenes

- **Unidad 2: Mejorando nuestra página**

Tablas
Formularios
Frames (marcos)
Sonido

- **Unidad 3: Páginas de estilo en Cascada y Javascript**

- Introducción a las CSS
- Características y ventajas de las CSS
- Usos de las CCS
- Reglas de importancia en los estilos
- Sintaxis CSS
- Atributos de las hojas de estilo
- Trucos avanzados con CSS
- Introducción a Javascript
- Algo de historia sobre Javascript
- Diferencias entre Java y Javascript
- Usos de Javascript
- Versiones de navegadores y de Javascript
- Efectos rápidos
- El lenguaje Javascript
- Sintaxis Javascript

UNIDAD 1

CONCEPTOS BÁSICOS**INTRODUCCIÓN**

Esta unidad busca que el estudiante distinga los conceptos básicos involucrados en la diseño y desarrollo de una página web, cual es la estructura básica de un documento HTML, como dar forma al texto del documento, como trabajar con imágenes y aprender a trabajar con los enlaces (hipervinculos).

JUSTIFICACIÓN

El fenómeno del web sigue creciendo a pasos gigantes. De la misma manera los profesionales involucrados en esta área deben, a la par de perfeccionar sus habilidades técnicas, planear eficazmente un proyecto de estas características.

El diseño de un lugar en Internet se determina a través de dos puntos de vista; por un lado está la planeación del sitio, su interactividad, su estructura de navegación y como será dispuesta al usuario. Por otra parte será la presentación de colores, gráficas, animaciones, imágenes y texto, factores que deben ser definidos en detalle. A lo largo de la unidad, mediante ejercicios prácticos, se aprenderán estos conceptos.

OBJETIVO GENERAL

Conocer el lenguaje de desarrollo de páginas web: HTML

OBJETIVOS ESPECÍFICOS

- Conocer los diferentes tipos de organización para un sitio Web
- Identificar la estructura básica de una página
- Conocer la manera de dar formato al texto de un documento Web
- Conocer como trabajar con Enlaces
- Aprender a trabajar con Imágenes en una página Web

CONTENIDO

1. Conceptos básicos
2. Organización de una WEB
3. Estructura de un desarrollo WEB
4. El lenguaje HTML
5. Editores y convertidores
6. Estructura básica de un documento HTML
7. Dando forma al texto del documento HTML
8. Enlaces
9. Imágenes

CONCEPTOS BÁSICOS

1. **World Wide Web (WWW):** Digamos, simplemente, que es un sistema de información, el sistema de información propio de Internet. Sus características son:
 - Información por hipertexto: Diversos elementos (texto o imágenes) de la información que se nos muestra en la pantalla están vinculados con otras informaciones que pueden ser de otras fuentes. Para mostrar en pantalla esta otra información bastará con hacer clic sobre ellos.
 - Gráfico: En la pantalla aparece simultáneamente texto, imágenes e incluso sonidos.
 - Global: Se puede acceder a él desde cualquier tipo de plataforma, usando cualquier navegador y desde cualquier parte del mundo.
 - Pública: Toda su información está distribuida en miles de ordenadores que ofrecen su espacio para almacenarla. Toda esta información es pública y toda puede ser obtenida por el usuario.

- **Dinámica:** La información, aunque esta almacenada, puede ser actualizada por el que la publica sin que el usuario deba actualizar su soporte técnico.
 - **Independiente:** Dada la inmensa cantidad de fuentes, es independiente y libre.
2. **Navegador:** Es el programa que nos ofrece acceso a Internet. Debe ser capaz de comunicarse con un servidor y comprender el lenguaje de todas las herramientas que manejan la información de Web. Puede decirse que cada casa de software podría tener su navegador propio, aunque los más populares sean Netscape e Internet Explorer.
 3. **Servidor:** Se encarga de proporcionar al navegador los documentos y medios que este solicita. Utiliza un protocolo HTTP para atender las solicitudes de archivos por parte de un navegador.
 4. **HTTP:** Es el protocolo de transferencia de hipertexto, o sea, el protocolo que los servidores de World Wide Web utilizan para mandar documentos HTML a través de Internet.
 5. **URL:** Es el Localizador Uniforme de Recursos, o dicho más claramente, es la dirección que localiza una información dentro de Internet.
 6. **HTML:** De momento, le basta saber que estas siglas se corresponden con la definición "Lenguaje para marcado de hipertexto". Más claro aún, se trata de un lenguaje para estructurar documentos a partir de texto en World Wide Web. Este lenguaje se basa en etiquetas (instrucciones que le dicen al texto como deben mostrarse) y atributos (parámetros que dan valor a la etiqueta).

ORGANIZACIÓN DE UNA WEB

Para hacer una buena presentación Web lo ideal es crearnos un boceto inicial de la estructura. Si hacemos esto, no solo estamos procurando una presentación agradable y facilitando la tarea de navegar sino que también nos facilitamos el mantenimiento de futuras revisiones y modificaciones.

1. Objetivos

Lo primero que debemos hacer es fijarnos los objetivos que queremos alcanzar según la información que vayamos a aportar. Para crear nuestra primera página, estos objetivos deberían no ser muy pretenciosos o tener un sentido únicamente personal. Tener claros los objetivos nos ayudará a no plasmar contenidos confusos o innecesarios.

2. Contenidos

Una vez tenemos los objetivos, hay que organizar el contenido por temas o secciones, que se ajusten a nuestros objetivos, reuniendo las informaciones relacionadas bajo el

mismo epígrafe. Es conveniente que los temas sean razonablemente cortos y si fuera necesario divida en subtemas. Si por el contrario tenemos temas muy cortos, lo correcto sería agruparlos bajo un encabezado de tema algo más general.

3. Primer paso

Una presentación Web consiste de una o más páginas Web que contienen texto y gráficos y que están vinculadas entre si creando un cuerpo de información. La página principal o página base es desde donde se comienza a visitar la presentación y su URL será la que figure como dirección de la presentación. Esta página base debe ofrecer un panorama general del contenido de la presentación.

4. Organización

Ha llegado la hora de estructurar la información recopilada en un conjunto de páginas Web. Podemos crearnos una estructura propia pero lo más lógico es guiarnos por una estructura clásica.

ESTRUCTURA DE UN DESARROLLO WEB

La estructura de un conjunto de páginas Web es muy importante, ya que una buena estructura permitirá al lector visualizar todos los contenidos de una manera fácil y clara, mientras que un conjunto de páginas Web con una mala estructura producirá en el lector una sensación de estar perdido, no encontrará rápidamente lo que busca y terminará por abandonar nuestro sitio.

Planifique la estructura antes de empezar

Antes de crear un conjunto de páginas Web uno ha de tener una idea clara de cómo va a ser la estructura de dichas páginas, es conveniente hacer algún esquema sencillo, para la mayoría de los casos una hoja de papel y un lapicero bastará, pero si el emplazamiento va a albergar un gran número de páginas es recomendable usar algún tipo de programa que permita manejar estructuras de tipo grafo.

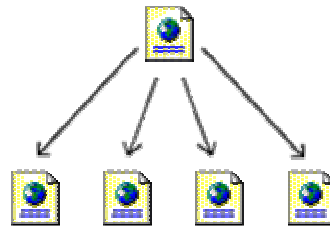
La estructura depende del contenido

No es lo mismo crear una estructura de navegación para un sitio que desea publicar información al estilo de un libro cuya estructura estará formada por capítulos, este tipo de información se adapta bastante bien a una estructura lineal como jerárquica. Mientras que un sitio donde se expone un tutorial o un tour es más apropiada una estructura de tipo lineal.

Tipos de Estructuras

1. Jerárquica

La estructura jerárquica, es la típica estructura de árbol, en el que la raíz es la hoja de bienvenida, esta hoja se puede también sustituir por la hoja de contenido, en la que se exponen las diferentes secciones que contendrá nuestro sitio. La selección de una sección nos conduce asimismo a una lista de subtemas que pueden o no dividirse. Este tipo de organización permite al lector conocer en qué lugar de la estructura se encuentra, además de saber que, con forme se adentra en la estructura obtiene información más específica y que la información más general se encuentra en los niveles superiores.

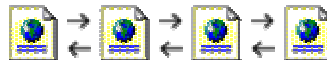


2. Lineal

La estructura lineal es la más simple de todas, la manera de recorrerla es la misma que si estuviésemos leyendo un libro, de manera que estando en una página, podemos ir a la siguiente página o a la anterior.

Esta estructura es muy útil cuando queremos que el lector siga un camino fijo y guiado, además impedimos que se distraiga con enlaces a otras páginas. Por otra parte podemos causar a lector la sensación de estar encerrado si el camino es muy largo o poco interesante.

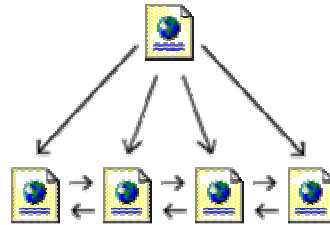
Este tipo de estructura sería válido para tutoriales de aprendizaje o tours de visita guiada.



3. Lineal con jerarquía

Este tipo de estructura es una mezcla de la dos anteriores, los temas y subtemas están organizados de una forma jerárquica, pero uno puede leer todo el contenido de una forma lineal si se desea.

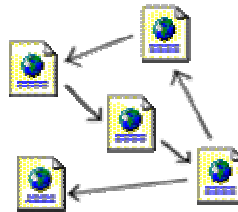
Esto permite tener el contenido organizado jerárquicamente y simultáneamente poder acceder a toda la información de una manera lineal como si estuviésemos leyendo un libro. Esta guía sigue básicamente este tipo de estructura.



4. Red

La estructura de red es una organización en la que aparentemente no hay ningún orden establecido, las páginas pueden apuntarse unas a otras sin ningún orden aparente.

Este tipo de organización es la más libre, pero también es la más peligrosa ya que si no se informa al lector de en dónde se encuentra, puede perderse o puede no encontrar lo que anda buscando o no llegar a ver lo que le queremos mostrar. Por eso es muy recomendable asociar la estructura de las páginas con alguna estructura conocida, como por ejemplo la de una ciudad.



5. Secuenciación

Consiste en decidir que contenido va en cada página, elaborar la trama de vínculos para navegar entre ellas e incluso, hacernos una idea de que tipo de gráficos vamos a poner y que ubicación van a tener. Para ello puede utilizarse un "Tablero de Secuencia", un esquema gráfico que nos ayudará a recordar en todo momento donde encaja cada página en el global de la presentación.

6. Revisión de objetivos

Finalmente y antes de ponernos a crear nuestra presentación Web, debemos prestar atención a que lo que tenemos plasmado en el "Tablero de Secuencia" cubre los objetivos que nos habíamos propuesto. Si es así, ya podemos comenzar a manejarnos con HTML.

EL LENGUAJE HTML

Como ya se ha dicho, este lenguaje estructura documentos. La mayoría de los documentos tienen estructuras comunes (títulos, párrafos, listas...) que van a ser definidas por este lenguaje mediante etiquetas. Cualquier cosa que no sea una etiqueta es parte del documento mismo.

Este lenguaje no describe la apariencia del diseño de un documento sino que ofrece a cada plataforma que le de formato según su capacidad y la de su navegador (tamaño de la pantalla, fuentes que tiene instaladas...). Por ello y para no frustrarnos, no debemos diseñar los documentos basándonos en como lucen en nuestro navegador sino que debemos centrarnos en proporcionar un contenido claro y bien estructurado que resulte fácil de leer y entender.

No se desespere por lo que acaba de leer. HTML tiene dos ventajas que lo hacen prácticamente imprescindibles a la hora de diseñar una presentación web: Su compatibilidad y su facilidad de aprendizaje debido al reducido número de etiquetas que usa.

Básicamente, los documentos escritos en HTML constan del texto mismo del documento y las etiquetas que pueden llevar atributos. Esto llevado a la práctica, vendría a ser:

<etiqueta> texto afectado </etiqueta>

La etiqueta del principio activa la orden y la última (que será la del principio precedida del signo /) la desactiva. No todas las etiquetas tienen principio y final pero esto lo veremos más adelante..

EDITORES Y CONVERTIDORES

Antes de comenzar al trabajar sobre un editor, le recomendaría que visionase el código fuente de una página WEB. Todos los navegadores dan la opción de editarla (Menú ver / Código fuente). Si visita otras páginas y visualiza su código fuente encontrará similitudes en la forma en que están organizadas las páginas y en las etiquetas utilizadas.

¿Dónde hay que editar el código fuente? Pues, si usted es usuario de Windows le bastaría con el Bloc de Notas y si utiliza Macintosh con el Simple Text. Si utiliza procesadores de texto más potentes debe guardar sus documentos como "solo texto" ya que HTML ignora todos los espacios en blanco. Una vez guardado convierta la extensión de texto por la extensión html o htm (en los sistemas DOS).

Los convertidores se utilizan para tomar los archivos de un procesador de textos y convertirlos a HTML. Pero debido a la propia limitación de este lenguaje, por muy elegante que hagamos un documento en nuestro procesador, un convertidor no obrará milagros y quizá acabe por crear cosas ilegibles en HTML. Además, la mayoría de los convertidores no convierten imágenes y no automatizan los vínculos hacia los documentos en Web debiendo corregir esto de manera manual.

A través de Internet o de revistas especializadas, usted podrá hacerse con editores y convertidores gratuitos o de muy reducidos costos.

Quizá más adelante, cuando este acostumbrado a trabajar con HTML, puedan resultarle interesantes pero eso se lo dejo a su futura elección. De momento, hágame caso, si quiere aprender HTML use solo un procesador de texto simple.

ESTRUCTURA BÁSICA DE UN DOCUMENTO HTML

El principio esencial del lenguaje HTML es el uso de las etiquetas (tags). Funcionan de la siguiente manera:

`<XXX>` Este es el inicio de una etiqueta.
`</XXX>` Este es el cierre de una etiqueta.

Las letras de la etiqueta pueden estar en mayúsculas o minúsculas, indiferentemente.

Lo que haya entre ambas etiquetas estará influenciada por ellas. Por ejemplo, todo el documento HTML debe estar entre las etiquetas `<html>` y `</html>`:

`<html>` [Todo el documento] `</html>`

Un documento HTML en sí está dividido en dos zonas principales:

- El encabezamiento, comprendido entre las etiquetas `<head>` y `</head>`
- El cuerpo, comprendido entre las etiquetas `<body>` y `</body>`

Dentro del encabezamiento hay información del documento, que no se ve en la pantalla principal del BROWSER que es utilizado para visualizar el documento HTML, principalmente la información encontrada en el encabezamiento es el título del documento, comprendido entre las etiquetas `<title>` y `</title>`. El título debe ser breve y descriptivo de su contenido, pues será lo que vean los demás cuando añadan nuestra página a su bookmark (o agenda de direcciones).

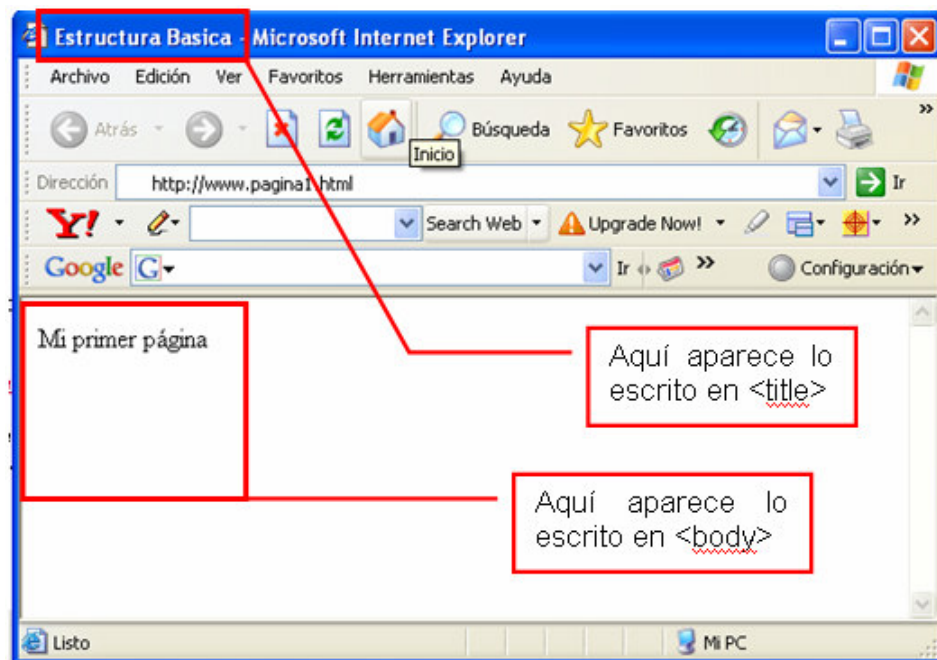
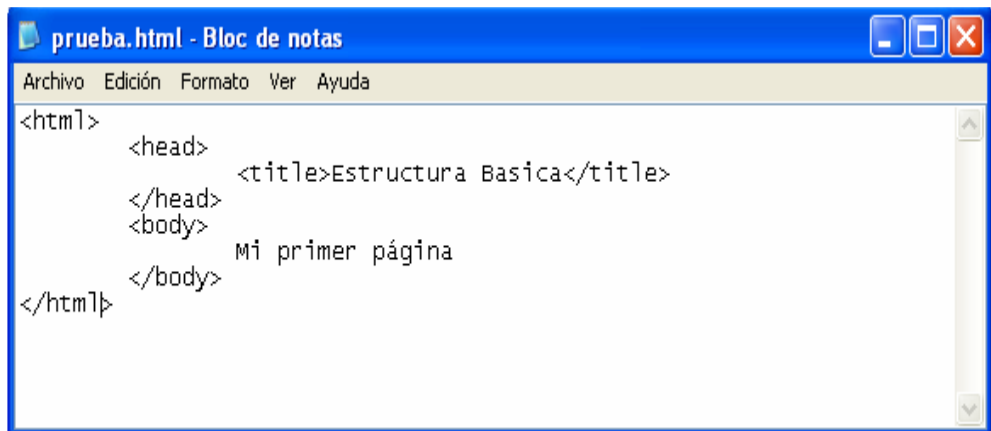
Dentro del cuerpo está todo lo que queremos que aparezca en la pantalla principal (texto, imágenes, etc.)

Por tanto, la estructura de un documento HTML queda de esta manera:

```

<html>
  <head>
    <title> Título de la página </title>
  </head>
  <body>
    [Aquí van las etiquetas que visualizan la página]
  </body>
</html>
    
```

Ejemplo:



NOTA: Las mayúsculas o minúsculas son indiferentes al escribir etiquetas

Las etiquetas pueden ser escritas con cualquier tipo de combinación de mayúsculas y minúsculas. <html>, <HTML> o <HtMl> son la misma etiqueta. Resulta sin embargo aconsejable acostumbrarse a escribirlas en minúscula ya que otras tecnologías que pueden convivir con nuestro HTML (XML por ejemplo) no son tan permisivas y nunca viene mal coger buenas costumbres desde el principio para evitar fallos triviales en un futuro.

DANDO FORMA AL TEXTO DEL DOCUMENTO HTML

Cuando escribimos en el documento el texto que queremos que aparezca en la pantalla, veremos que éste se acomoda a ella, sin que tengamos que pulsar el retorno del carro. Formatear un texto pasa por tareas tan evidentes como definir los párrafos, justificarlos, introducir viñetas, numeraciones o bien poner en negrita, itálica. Para dar formato al texto iniciemos viendo algunas etiquetas.

<p> ... </p>

Párrafos: Esta etiqueta, en un principio, se diseñó para saltar de párrafo por lo que puede ir sola "<P>" al final de un texto indicando que a continuación se quiere una línea en blanco

**
**

Salto de línea. Esta etiqueta sirve para realizar un salto de línea, puede poner tantas como desee y realizará un salto de línea por cada una de ellas.

<!-- ... -->

Comentarios. Son directivas que nunca se mostrarán a través del navegador y que le servirán para recordatorios en futuras revisiones del documento

<h1> ... </h1>

Titulares. Sirven para dividir el texto en secciones. Se pueden definir seis niveles de titulares, el texto que deseamos que sea un titular se pone entre las etiquetas <h1> Titular </h1>. Se definen mediante las etiquetas <h1>.....</h1> hasta <h6>.....</h6>

<center> ... </center>

Centrar: Nos centra todo lo que esté dentro de ella, ya sea texto, imágenes, etc.

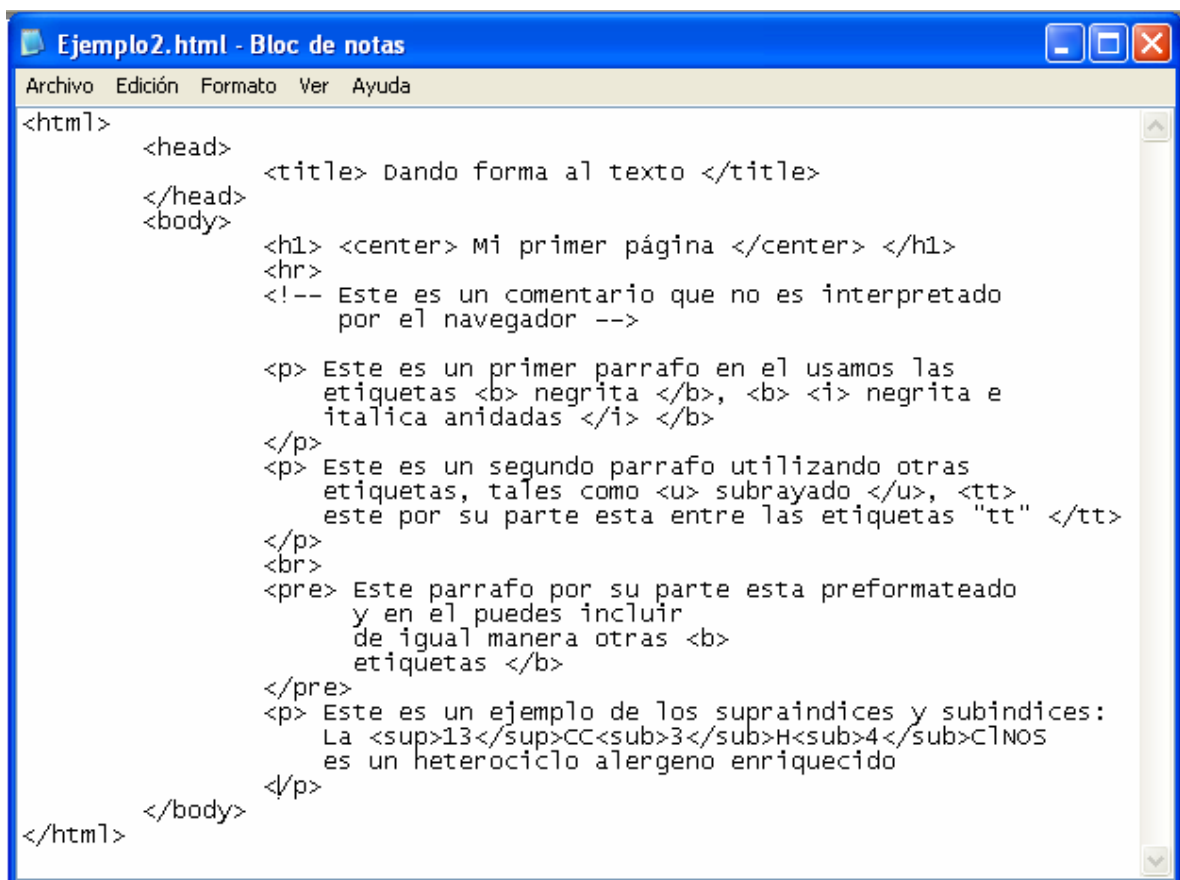
<code><hr></code>	Separadores (horizontal rules), que se consiguen con la etiqueta <code><hr></code> (no existe la correspondiente de cierre). Con ella se obtiene una raya horizontal tan ancha como la pantalla, y con la apariencia de estar embutida sobre el fondo
<code> ... </code>	Negrita: Poner un texto en negrita
<code><i > ... </i ></code>	Itálica: Poner un texto en itálica (cursiva)
<code><u> ... </u></code>	Subrayado: (Underlined): Subraya el texto.
<code><sup> ... </sup></code>	Supraíndices
<code><sub> ... </sub></code>	Subíndices
<code><pre> ... </pre></code>	Preformateado: El texto que se encuentre entre ella estará preformateado, es decir, que aparecerá como si hubiera sido escrito con una máquina de escribir, con una fuente de espaciado fijo (tipo Courier). Además se respetarán los espacios en blanco y retornos del carro. Es muy apropiada para confeccionar tablas y otros documentos similares.
<code><tt> ... </tt></code>	Typewriter: conseguimos también que el texto tenga un tamaño menor y la apariencia de los caracteres de una máquina de escribir. La diferencia con la anterior es que no preformatea el texto, sino que únicamente cambia su apariencia.
<code><blockquote></code> <code></blockquote></code>	... Se utiliza para destacar una cita textual dentro del texto general

Anidar etiquetas

Todas estas etiquetas y por supuesto el resto de las que veremos más adelante pueden ser anidadas unas dentro de otras y de tal manera conseguir resultados diferentes. Así, podemos sin ningún problema crear texto en negrita e itálica embebiendo una etiqueta dentro de la otra:

Consejo: Cuando anides etiquetas HTML hazlo correctamente. Es decir, que si abres etiquetas dentro de otra (principal), antes de cerrar la etiqueta principal cierras las etiquetas que hayas abierto dentro de ella

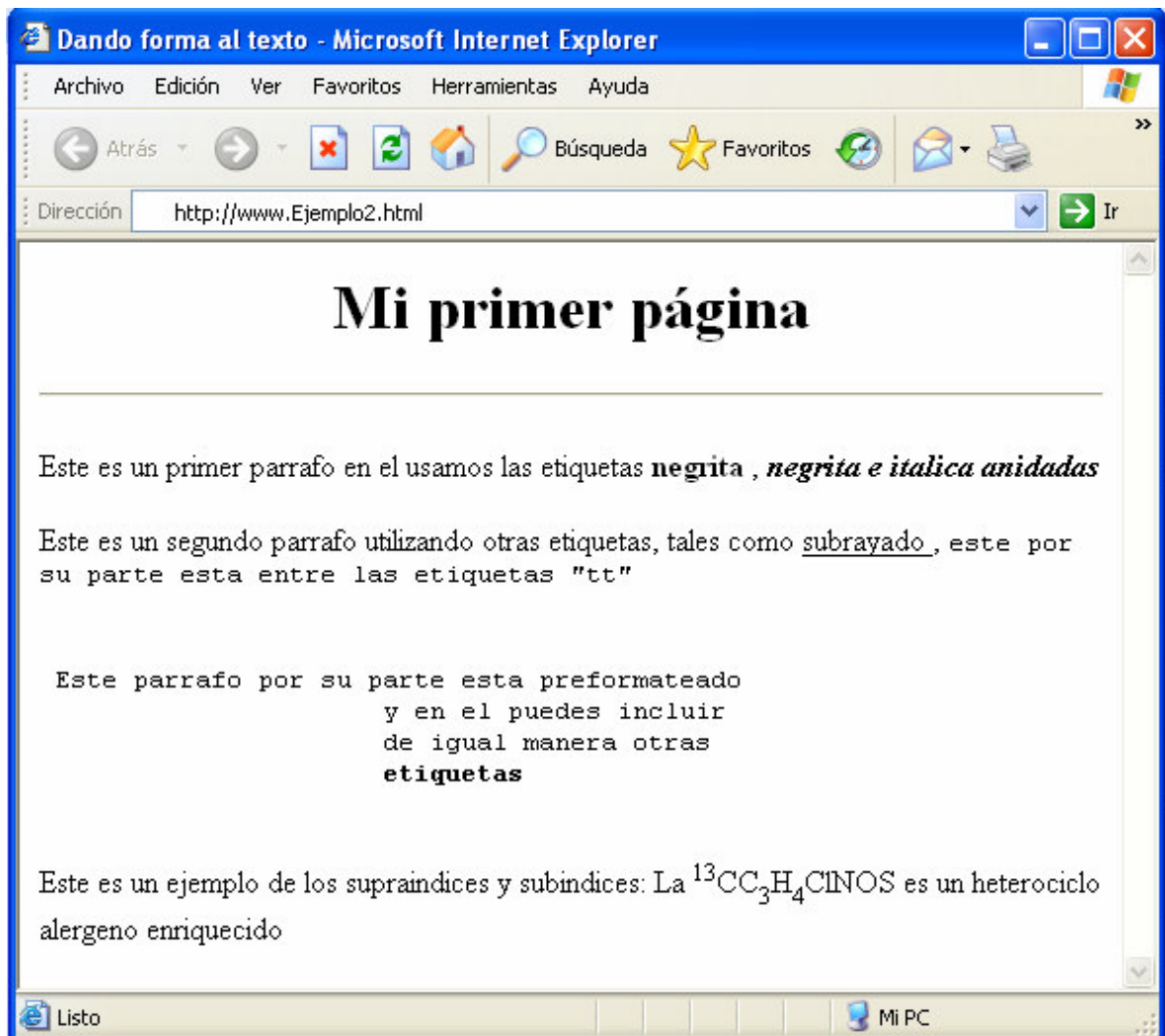
Ejemplo

A screenshot of a Notepad window titled 'Ejemplo2.html - Bloc de notas'. The window has a menu bar with 'Archivo', 'Edición', 'Formato', 'Ver', and 'Ayuda'. The main text area contains the following HTML code:

```
<html>
  <head>
    <title> Dando forma al texto </title>
  </head>
  <body>
    <h1> <center> Mi primer página </center> </h1>
    <hr>
    <!-- Este es un comentario que no es interpretado
         por el navegador -->

    <p> Este es un primer párrafo en el usamos las
        etiquetas <b> negrita </b>, <b> <i> negrita e
        itálica anidadas </i> </b>
    </p>
    <p> Este es un segundo párrafo utilizando otras
        etiquetas, tales como <u> subrayado </u>, <tt>
        este por su parte esta entre las etiquetas "tt" </tt>
    </p>
    <br>
    <pre> Este párrafo por su parte esta preformateado
        y en el puedes incluir
        de igual manera otras <b>
        etiquetas </b>
    </pre>
    <p> Este es un ejemplo de los supraíndices y subíndices:
        La <sup>13</sup>CC<sub>3</sub>H<sub>4</sub>C1NOS
        es un heterociclo alergeno enriquecido
    </p>
  </body>
</html>
```

El resultado en el navegador será:



Los párrafos delimitados por etiquetas `<p>` pueden ser fácilmente justificados a la izquierda, centro o derecha especificando dicha justificación en el interior de la etiqueta por medio de un atributo `align`. Un atributo no es más que un parámetro incluido en el interior de la etiqueta que ayuda a definir el funcionamiento de la etiqueta de una forma más personal.

Así, si deseásemos introducir un texto alineado a la izquierda escribiríamos:

```
<p align="left">Texto alineado a la izquierda</p>
```

Para una justificación al centro:

```
<p align="center">Texto alineado al centro</p>
```

Para justificar a la derecha:

```
<p align="right">Texto alineado a la derecha</p>
```

En cada caso el atributo *align* toma determinados valores que son escritos entre comillas. En algunas ocasiones necesitamos especificar algunos atributos para el correcto funcionamiento de la etiqueta. En otros casos, el propio navegador toma un valor definido por defecto. Para el caso de *align*, el valor por defecto es *left*.

Nota: Los atributos tienen sus valores indicados entre comillas ("), pero si no los indicamos entre comillas también funcionará en la mayoría de los casos. Sin embargo, es aconsejable que pongamos siempre las comillas para acostumbrarnos a utilizarlas, por dar homogeneidad a nuestros códigos y para evitar errores futuros en sistemas más quisquillosos

El atributo *align* no es exclusivo de la etiqueta `<p>`. Otras etiquetas muy comunes, que veremos más adelante, entre las cuales se introducen texto o imágenes, suelen hacer uso de este atributo de una forma habitual.

Si tenemos un texto relativamente largo donde todos los párrafos están alineados a la izquierda (por ejemplo). Una forma de simplificar nuestro código y de evitar introducir continuamente el atributo *align* sobre cada una de nuestras etiquetas es utilizando la etiqueta `<div>`.

Esta etiqueta por sí sola no sirve para nada. Tiene que estar acompañada del atributo *align* y lo que nos permite es alinear cualquier elemento (párrafo o imagen) de la manera que nosotros deseemos.

Así, el código:

```
<p align="left"> Parrafo1</p>
<p align="left"> Parrafo3</p>
<p align="left"> Parrafo2</p>
```

es equivalente a:

```
<div align="left">
<p>Parrafo1</p>
<p>Parrafo2</p>
<p>Parrafo3</p>
</div>
```

Como se ha visto, la etiqueta `<div>` marca divisiones en las que se define un mismo tipo de alineado.

ACTIVIDAD...

Para practicar un poco lo que acabamos de ver vamos a realizar un ejercicio, en el que simplemente queremos construir una página que tenga, por este orden:

3 Párrafos alineados a la izquierda

2 Párrafos centrados

Un salto de línea doble

1 párrafo alineado a la derecha

Color, tamaño y tipo de letra

A pesar de que por razones de homogeneidad y sencillez de código este tipo de formatos son controlados actualmente **por hojas de estilo en cascada** (las cuales veremos mas adelante), existe una forma clásica y directa de definir color tamaño y tipo de letra de un texto determinado.

Esto se hace a partir de la etiqueta `` y su cierre correspondiente. Dentro de esta etiqueta deberemos especificar los atributos correspondientes a cada uno de estos parámetros que deseamos definir, tales como:

Atributo face: Define el tipo de letra. Este atributo es interpretado por versiones de Netscape a partir de la 3 y de MSIE 3 o superiores. Otros navegadores las ignoran completamente y muestran el texto con la fuente que utilizan.

Hay que tener cuidado con este atributo ya que cada usuario, dependiendo de la plataforma que utilice, puede no disponer de los mismos tipos de letra que nosotros con lo que, si nosotros elegimos un tipo del que no dispone, el navegador se verá forzado a mostrar el texto con la fuente que utiliza por defecto (suele ser Times New Roman). Para evitar esto, dentro del atributo suelen seleccionarse varios tipos de letra separados por comas. En este caso el navegador comprobará que dispone del primer tipo enumerado y si no es así, pasará al segundo y así sucesivamente hasta encontrar un tipo que posea o bien acabar la lista y poner la fuente por defecto. Veamos un ejemplo.

```
<font face="Comic Sans MS,arial,verdana">Este texto tiene otra tipografía</font>
```

Atributo size: Define el tamaño de la letra. Este tamaño puede ser absoluto o relativo.

Si hablamos en términos absolutos, existen 7 niveles de tamaño distintos numerados de 1 a 7 por orden creciente. Elegiremos por tanto un valor `size="1"` para la letra más pequeña o `size="7"` para la más grande.

Este texto es más grande

Podemos asimismo modificar el tamaño de nuestra letra con respecto al del texto mostrado precedentemente definiendo el número de niveles que queremos subir o bajar en esta escala de tamaños por medio de un signo + o -. De este modo, si definimos nuestro atributo como `size="+1"` lo que queremos decir es que aumentamos de un nivel el tamaño de la letra. Si estábamos escribiendo previamente en 3, pasaremos automáticamente a 4.

Los tamaños reales que veremos en pantalla dependerán de la definición y del tamaño de fuente elegido por el usuario en el navegador. Este tamaño de fuente puede ser definido en el Explorer yendo al menú superior, Ver/Tamaño de la fuente. En Netscape elegiremos View/Text Size. Esta flexibilidad puede en más de una ocasión resultarnos embarazosa ya que en muchos casos desearemos que el tamaño del texto permanezca constante para que éste quepa en un determinado espacio. Veremos en su momento que esta prefijación del tamaño puede ser llevada a cabo por las hojas de estilo en cascada.

Atributo color: El color del texto puede ser definido mediante el atributo color. Cada color es a su vez definido por un número hexadecimal que está compuesto a su vez de tres partes. Cada una de estas partes representa la contribución del rojo, verde y azul al color en cuestión.

Por otra parte, es posible definir de una manera inmediata algunos de los colores más frecuentemente usados para los que se ha creado un nombre más nemotécnico:

Nombre Color

Aqua	
Black	
Blue	
Fuchsia	
Gray	
Green	
Lime	
Maroon	
Navy	
Olive	
Purple	
Red	
Silver	

Teal 
White
Yellow 

Este texto está en rojo

ACTIVIDAD...

*Pongamos pues en practica todo lo que hemos aprendido haciendo un ejercicio consistente en una página que tenga las siguientes características:
Un titular con encabezado de nivel 1, en itálica y color verde oliva.
Un segundo titular con encabezado de nivel 2, también de color verde oliva.
Todo el texto de la página deberá presentarse con una fuente distinta de la fuente por defecto. Por ejemplo "Comic Sans MS" y en caso de que ésta no esté en el sistema que se coloque la fuente "Arial".*

NOTA: Utilice la mayor de etiquetas posibles, de las vistas hasta ahora.

Atributos para páginas

Las páginas HTML pueden construirse con variedad de atributos que le pueden dar un aspecto a la página muy personalizado. Podemos definir atributos como el color de fondo, el color del texto o de los enlaces. Estos atributos se definen en la etiqueta <body> y, como decíamos son generales a toda la página.

- **Atributos para fondos**

bgcolor: especificamos un color de fondo para la página. El color de fondo que podemos asignar con bgcolor es un color plano, es decir el mismo para toda la superficie del navegador.

background: Sirve para indicar la colocación de una imagen como fondo de la página. La imagen se coloca haciendo un mosaico, es decir, se repite muchas veces hasta ocupar todo el espacio del fondo de la página. Más adelante veremos como se insertan imágenes con HTML y los tipos de imágenes que se pueden utilizar.

Consejo: siempre que coloquemos una imagen de fondo, debemos poner también un color de fondo cercano al color de la imagen.

Esto se debe a que, al colocar una imagen de fondo, el texto de la página debemos colocarlo en un color que contraste suficientemente con dicho fondo. Si el visitante no puede ver el fondo por cualquier cuestión (Por ejemplo tener deshabilitada la carga de imágenes) puede que el texto no contraste lo suficiente con el color de fondo por defecto de la web.

Creo que lo mejor será poner un ejemplo. Si la imagen de fondo es oscura, tendremos que poner un texto claro para que se pueda leer. Si el visitante que accede a la página no ve la imagen de fondo, le saldrá el fondo por defecto, que generalmente es blanco, de modo que al tener un texto con color claro sobre un fondo blanco, nos pasará que no podremos leer el texto convenientemente.

Ocurre parecido cuando se está cargando la página. Si todavía no ha llegado a nuestro sistema la imagen de fondo, se verá el fondo que hayamos seleccionado con bgcolor y es interesante que sea parecido al color de la imagen para que se pueda leer el texto mientras se carga la imagen de fondo.

- **Color del texto:**

text: este atributo sirve para asignar el color del texto de la página. Por defecto es el negro.

Además del color del texto, tenemos tres atributos para asignar el color de los enlaces de la página. Ya debemos saber que los enlaces deben diferenciarse del resto del texto de la página para que los usuarios puedan identificarlos fácilmente. Para ello suelen aparecer subrayados y con un color más vivo que el texto. Los tres atributos son los siguientes:

link: el color de los enlaces que no han sido visitados. (por defecto es azul clarito)

vlink: el color de los enlaces visitados. La "v" viene justamente de la palabra visitado. Es el color que tendrán los enlaces que ya hemos visitado. Por defecto su color es morado. Este color debería ser un poco menos vivo que el color de los enlaces normales.

alink: es el color de los enlaces activos. Un enlace está activo en el preciso instante que se pulsa. A veces es difícil darse cuenta cuando un enlace está activo porque en el momento en el que se activa es porque lo estamos pulsando y en ese caso el navegador abandonará la página rápidamente y no podremos ver el enlace activo más que por unos instantes mínimos.

- **Márgenes:** Con otros atributos de la etiqueta <body> se pueden asignar espacios de margen en las páginas, lo que es muy útil para eliminar los márgenes en blanco que aparecen a los lados, arriba y debajo de la página. Estos atributos son distintos para Internet Explorer y para Netscape Navigator, por lo que debemos utilizarlos todos si queremos que todos los navegadores los interpreten perfectamente.

leftmargin: para indicar el margen a los lados de la página. Válido para iexplorer.

topmargin: para indicar el margen arriba y debajo de la página. Para iexplorer.

marginwidth: la contrapartida de leftmargin para Netscape. (Margen a los lados)

marginheight: igual que topmargin, pero para Netscape. (Margen arriba y abajo)

Ejemplo:

```
<body bgcolor="#000000" text="#ffffff" link="#ffff33" alink="#ffffcc"
alink="#ffff00">
```

Listas

Las posibilidades que nos ofrece el HTML en cuestión de tratamiento de texto son realmente notables. No se limitan a lo visto hasta ahora, sino que van más lejos todavía. Varios ejemplos de ello son las listas, que sirven para enumerar y definir elementos, los textos preformateados y las cabeceras o títulos.

Las listas son utilizadas para citar, numerar y definir objetos. También son utilizadas corrientemente para desplazar el comienzo de línea hacia la derecha.

Podemos distinguir tres tipos de listas:

- Listas desordenadas
- Listas ordenadas
- Listas de definición

- **Listas desordenadas**

Son delimitadas por las etiquetas y (unordered list). Cada uno de los elementos de la lista es citado por medio de una etiqueta (sin cierre, aunque no hay inconveniente en colocarlo).

Ejemplo:

```
<p>Países del mundo</p>
<ul>
  <li>Argentina
  <li>Perú
  <li>Chile
```

```
</ul>
```

Podemos definir el tipo de viñeta empleada para cada elemento. Para ello debemos especificarlo por medio del atributo `type` incluido dentro de la etiqueta de apertura ``, si queremos que el estilo sea válido para toda la lista, o dentro de la etiqueta `` si queremos hacerlo específico de un solo elemento. La sintaxis es:

```
<ul type="tipo de viñeta">
```

donde tipo de viñeta puede ser uno de los siguientes:

```
circle  
disc  
square
```

Nota: En algunos navegadores no funciona la opción de cambiar el tipo de viñeta a mostrar y por mucho que nos empeñemos, siempre saldrá el redondel negro.

En caso de que no funcione siempre podemos construir la lista a mano con la viñeta que queramos utilizando las tablas de HTML. Veremos más adelante cómo trabajar con tablas.

Vamos a ver un ejemplo de lista con un cuadrado en lugar de un redondel, y en el último elemento colocaremos un círculo. Para ello vamos a colocar el atributo `type` en la etiqueta ``, con lo que afectará a todos los elementos de la lista.

```
<ul type="square">  
<li>Elemento 1  
<li>Elemento 2  
<li>Elemento 3  
<li type="circle">Elemento 4  
</ul>
```

- **Listas ordenadas**

En este caso usaremos las etiquetas `` (ordered list) y su cierre. Cada elemento será igualmente precedido de su etiqueta ``.

Ejemplo:

```
<p>Reglas de comportamiento en el trabajo</p>  
<ol>  
<li>El jefe siempre tiene la razón  
<li>En caso de duda aplicar regla 1
```


Del mismo modo que para las listas desordenadas, las listas ordenadas ofrecen la posibilidad de modificar el estilo. En concreto nos es posible especificar el tipo de numeración empleado eligiendo entre números (1, 2, 3...), letras (a, b, c...) y sus mayúsculas (A, B, C,...) y números romanos en sus versiones mayúsculas (I, II, III,...) y minúsculas (i, ii, iii,...).

Para realizar dicha selección hemos de utilizar, como para el caso precedente, el atributo `type`, el cual será situado dentro de la etiqueta ``. Los valores que puede tomar el atributo en este caso son:

- | | |
|---|--|
| 1 | Para ordenar por números |
| a | Por letras del alfabeto |
| A | Por letras mayúsculas del alfabeto |
| i | Ordenación por números romanos en minúsculas |
| I | Ordenación por números romanos en mayúsculas |

Puede que en algún caso deseemos comenzar nuestra enumeración por un número o letra que no tiene por qué ser necesariamente el primero de todos. Para solventar esta situación, podemos utilizar un segundo atributo, `start`, que tendrá como valor un número. Este número, que por defecto es 1, corresponde al valor a partir del cual comenzamos a definir nuestra lista. Para el caso de las letras o los números romanos, el navegador se encarga de hacer la traducción del número a la letra correspondiente.

Ejemplo:

```
<p>Ordenamos por numeros</p>
<ol type="1">
<li>Elemento 1
<li> Elemento 2
</ol>
```

```
<p>Ordenamos por letras</p>
<ol type="a">
<li>Elemento a
<li> Elemento b
</ol>
```

```
<p>Ordenamos por números romanos empezando por el 10</p>
<ol type="i" start="10">
<li>Elemento x
<li> Elemento xi
</ol>
```

- **Listas de definición**

Cada elemento es presentado junto con su definición. La etiqueta principal es <dl> y </dl> (definition list). La etiquetas del elemento y su definición son <dt> (definition term) y <dd> (definition definition) respectivamente.

Ejemplo:

```
<p>Diccionario de la Real Academia</p>
<dl>
  <dt>Brujula
  <dd>Señórua montada en una escóbula
  <dt>Oreja
  <dd>Sesenta minutejos
</dl>
```

Cada línea <dd> esta desplazada hacia la izquierda. Este tipo de etiquetas son usadas a menudo con el propósito de crear textos más o menos desplazados hacia la izquierda.

Ejemplo:

```
<dl>
<dd>Primer nivel de desplazamiento
  <dl>
    <dd>Segundo nivel de desplazamiento
      <dl>
        <dd>Tercer nivel de desplazamiento
      </dl>
    </dl>
  </dl>
</dl>
```

- **Anidando listas**

Nada nos impide utilizar todas estas etiquetas de forma anidada como hemos visto en otros casos. De esta forma, podemos conseguir listas mixtas como por ejemplo:

```
<p>Ciudades del mundo</p>
<ul>
  <li>Argentina
    <ol>
      <li>Buenos Aires
      <li>Bariloche
    </ol>
  <li>Uruguay
  <ol>
```

```

        <li>Montevideo
        <li>Punta del Este
    </ol>
</ul>

```

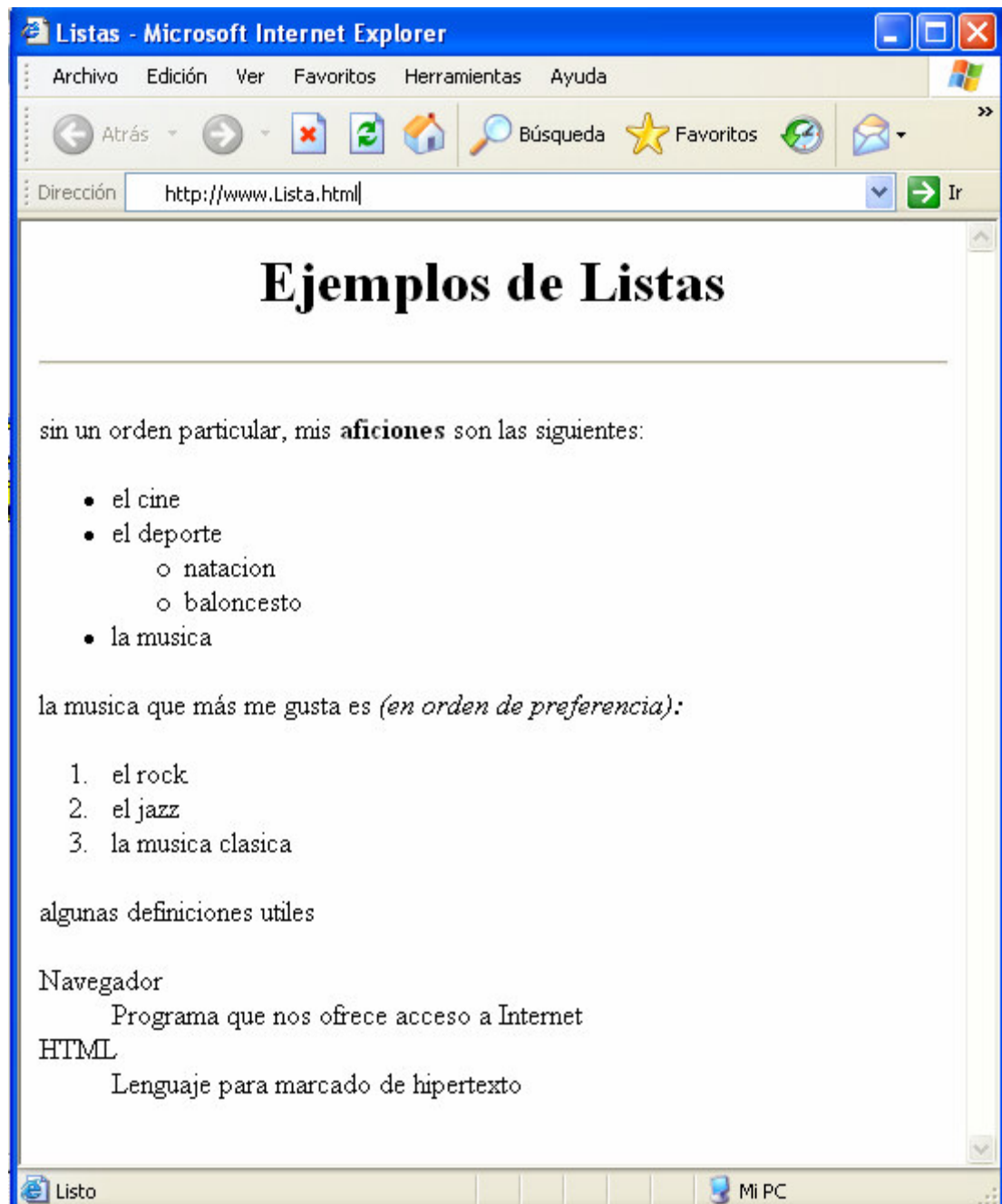
Ejemplo:

```

<html>
  <head>
    <title> Listas </title>
  </head>
  <body>
    <center>
      <h1> Ejemplos de Listas </h1>
    </center>
    <hr>
    <p>sin un orden particular, mis <b> aficiones </b> son las siguientes:
    </p>
    <!--una lista sin orden alguno -->
    <ul>
      <li> el cine
      <li> el deporte
    </ul>
      <li> natacion
      <li> baloncesto
    </ul>
      <li> la musica
    </ul>
    <p>la musica que más me gusta es <i> (en orden de preferencia): </i>
    </p>
    <!--una lista con un orden -->
    <ol>
      <li> el rock
      <li> el jazz
      <li> la musica clasica
    </ol>
    <p>algunas definiciones utiles
    </p>
    <dl>
      <dt>Navegador
      <dd>Programa que nos ofrece acceso a Internet
      <dt>HTML
      <dd>Lenguaje para marcado de hipertexto
    </dl>
  </body>
</html>

```

El resultado será:



ACTIVIDAD...

Elabore un página que contenga los tres tipos de listas (Ordenadas, Sin orden y de definición). Las listas de definición deben estar anidadas dentro de la lista ordenada.

Caracteres especiales

Una página web se ha de ver en países distintos, que usan conjuntos de caracteres distintos. El lenguaje HTML nos ofrece un mecanismo por el que podemos estar seguros que una serie de caracteres raros se van a ver bien en todos los ordenadores del mundo, independientemente de su juego de caracteres.

Este conjunto son los caracteres especiales. Cuando queremos poner uno de estos caracteres en una página, debemos sustituirlo por su código.

Por ejemplo, la "á" (a minúscula acentuada) se escribe "´" de modo que la palabra página se escribiría en una página HTML de este modo: p&aamp;acute;gina

Caracteres especiales básicos

En realidad estos caracteres se usan en HTML para no confundir un principio o final de etiqueta, unas comillas o un & con su correspondiente caracter.

<	<	>	>
&	&	"	"

Caracteres especiales del HTML 2.0

Á	Á	À	À
É	É	È	È
Í	Í	Ì	Ì
Ó	Ó	Ò	Ò
Ú	Ú	Ù	Ù
á	á	à	à
é	é	è	è
í	í	ì	ì
ó	ó	ò	ò
ú	ú	ù	ù
Ä	Ä	Â	Â

Ë	Ë	Ê	Ê
&luml;	Ĭ	Î	Î
Ö	Ö	Ô	Ô
Ü	Ü	Û	Û
ä	ä	â	â
ë	ë	ê	ê
ï	ï	î	î
ö	ö	ô	ô
ü	ü	û	û
Ã	Ã	å	å
Ñ	Ñ	Å	Å
Õ	Õ	Ç	Ç
ã	ã	ç	ç
ñ	ñ	Ý	Ý
õ	õ	ý	ý
Ø	Ø	ÿ	ÿ
ø	ø	Þ	Þ
Ð	Ð	þ	þ
ð	ð	Æ	Æ
ß	ß	æ	æ

Caracteres especiales del HTML 3.2

¼	¼	 	
½	½	¡	!
¾	¾	£	£
©	©	¥	¥
®	®	§	§
ª	ª	¤	¤
²	²	¦	
³	³	«	«
¹	¹	¬	¬
¯	¯	­	–
µ	µ	º	º
¶	¶	´	´
·	·	¨	¨
°	°	±	±

¸	ç	»	»
¿	¿		

Otros caracteres especiales

×	×	¢	¢
÷	÷	€	€
“	“	™	™
”	”	‰	%o
Œ	Œ	ƒ	f
‡	‡	†	†

ENLACES

Lo característico del lenguaje HTML es el poder generar vínculos de hipertexto para enlazar con ellos todos sus documentos en web.

Para generar un enlace a otro documento necesitamos el nombre de un archivo (o su dirección URL) y el texto que servirá de punto de activación del otro documento. Este segundo elemento será el que veamos en pantalla y que se servirá del primero para saltar de documento.

Los enlaces se generan mediante la etiqueta `<a>.....` y, a diferencia de los vistos anteriormente, llevará siempre dentro de la etiqueta un atributo ya sea `` o ``

En general, los enlaces tienen la siguiente estructura:

```
<a href = "XXX"> YYY </a>
```

Donde XXX es el destino del enlace (Obsérvese las comillas). YYY es el texto indicativo en la pantalla del enlace (con un color especial y generalmente subrayado)

Tipos de enlaces

- Enlaces dentro de la misma página
- Enlaces con otra página nuestra
- Enlaces con una página fuera de nuestro sistema

- Enlaces con una dirección de e-mail
- Enlaces con archivos: para que los usuarios puedan descargarlos

- **Enlaces dentro de la misma página**

A veces, en el caso de documentos (o páginas) muy extensos, nos puede interesar dar un salto desde una posición a otra determinada. En este caso, lo que antes hemos llamado XXX, es decir, el destino del enlace, en este caso el sitio dentro de la página a donde queremos saltar, se sustituye por #MARCA (la palabra MARCA puede ser cualquier palabra que queramos). Lo que hemos llamado antes YYY es la palabra (o palabras) que aparecerán en la pantalla en color (en forma de hipertexto). Su estructura es, entonces:

```
<a href="#MARCA"> YYY </a>
```

Y en el sitio exacto a donde queremos saltar, debemos poner la siguiente etiqueta:

```
<a name ="MARCA"> YYY </a>
```

- **Enlaces con otra página nuestra**

Puede ser que tengamos una sola página. Pero lo más frecuente es que tengamos varias páginas, una inicial (o principal) y otras conectadas a ella, e incluso entre ellas mismas.

Supongamos que queremos enlazar con una de nuestras páginas llamada "contactos.html". En este caso, simplemente sustituimos lo que hemos llamado XXX (el destino del enlace) por el nombre del archivo:

```
<a href="contactos.html"> si quiere contactarse con nosotros </a>
```

Si queremos que vaya a un sitio concreto de otra página nuestra en vez de ir al principio de la página, adonde va por defecto, en ese sitio tenemos que colocar una marca (ver la Enlaces dentro de la misma página), y completar el enlace con la referencia a esa marca.

Lo veremos con el siguiente ejemplo: es la marca que colocaremos en nuestra pagina, que deseamos accesar desde otra nuestra. Entonces la etiqueta tiene que ser: En mi otra pagina .

Nota: Pudiera ocurrir que nuestro sitio WEB esté organizado con un directorio principal, y otros subdirectorios auxiliares. Si la página a la que deseamos acceder está, por ejemplo en el subdirectorio misubdir, entonces en la etiqueta tendría que colocarse "misubdir/mipag2.html".

Y a la inversa, si quiero saltar desde una página a otra que está en un directorio anterior, en la etiqueta tendría que haber puesto "../mipag2.html". Esos dos puntos hace que se dirija al directorio anterior. Obsérvese que se debe utilizar el símbolo / para indicar los subdirectorios, y no este otro \, que es propio únicamente de Windows.

Si nos queremos evitar todas estas complicaciones, podemos tener todo junto en un único directorio, pero esto tiene el inconveniente de que esté todo más desordenado, y sean más difíciles de hacer las futuras modificaciones..

- **Enlaces con una página fuera de nuestro sistema**

Si queremos enlazar con una página que esté fuera de nuestro sistema (es decir, que esté en un servidor distinto al que soporta nuestra página), es necesario conocer su dirección completa, o URL (Uniform Resource Locator). El URL podría ser, además de la dirección de una página del WEB, una dirección de FTP, GOPHER, etc.

Una vez conocida la dirección (o URL), lo colocamos en vez de lo que hemos llamado anteriormente XXX (el destino del enlace). Si queremos enlazar por ejemplo con la página de Netscape (cuyo URL es: <http://home.netscape.com/>), la etiqueta sería:

```
<a href="http://home.netscape.com/"> Página inicial de Netscape </a>
```

Es muy importante copiar estas direcciones correctamente (respetando las mayúsculas y minúsculas, pues los servidores UNIX sí las distinguen)

- **Enlaces con una dirección de e-mail**

En este caso, sustituimos lo que se ha llamado antes XXX (el destino del enlace) por mailto: seguido de la dirección de e-mail. La estructura de la etiqueta es:

```
<a href="mailto: dirección de e-mail"> Texto del enlace </a>
```

Un ejemplo podría ser:

```
<a href="mailto: jjmr13@gmail.com"> John Monsalve </a>
```

Hay algunos navegadores que no subrayan el comentario de este tipo de enlace.

Una manera recomendable y más segura para conocer la dirección e-mail sería poner algo así como:

Comentarios a John Jairo Monsalve en jjmr13@gmail.com

Es decir, es conveniente, por la razón dicha anteriormente, poner también en el texto del enlace la dirección de e-mail.

- **Enlaces con un archivo**

Este no es un tipo de enlace propiamente dicho, pero lo señalamos aquí porque son un tipo de enlaces muy habitual y que presenta alguna complicación para el usuario novato.

El mecanismo es el mismo que hemos conocido en los enlaces locales y los enlaces remotos, con la única particularidad de que en vez de estar dirigidos hacia una página web está dirigido hacia un archivo de otro tipo.

Si queremos enlazar con un archivo `mi_archivo.zip` que se encuentra en el mismo directorio que la página se escribiría un enlace así.

```
<a href="mi_archivo.zip">Descarga mi_archivo.zip</a>
```

Si pinchamos un enlace de este tipo nuestro navegador descargará el archivo, haciendo la pregunta típica de "Qué queremos hacer con el archivo. Abrirlo o guardarlo en disco".

Si queremos enlazar hacia otro tipo de archivo como un PDF o un mundo VRML (Realidad virtual para Internet) lo seguimos haciendo de la misma manera. El navegador, si reconoce el tipo de archivo, es el responsable de abrirlo utilizando el conector adecuado para ello. Así, si por ejemplo enlazamos con un PDF pondrá el programa Acrobat Reader en funcionamiento para mostrar los contenidos. Si enlazamos con un mundo VRML pondrá en marcha el plug-in que el usuario tenga instalado para ver los mundos virtuales (Cosmo Player por ejemplo).

Este sería un ejemplo de enlace a un documento PDF.

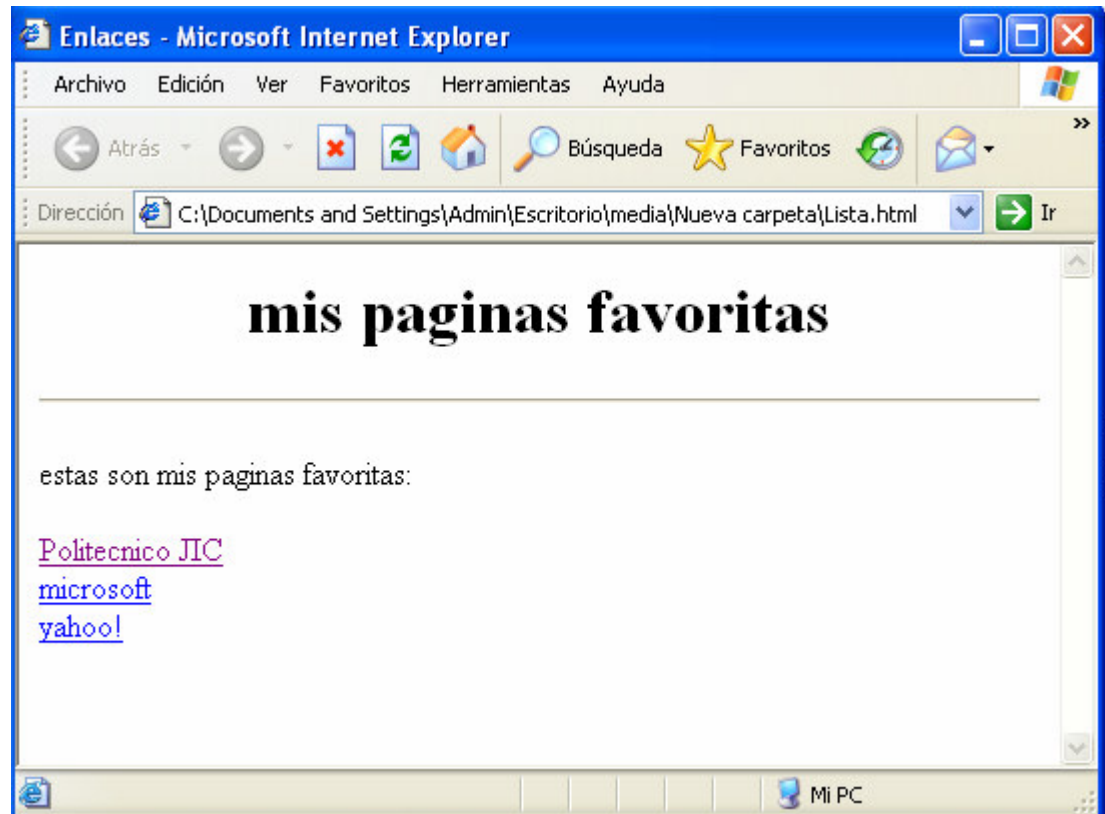
```
<a href="mi_documento_pdf">Descarga el PDF</a>
```

Ejemplo:

```
<html>
  <head>
    <title> Enlaces </title>
  </head>
```

```
<body>
  <center>
    <h1> mis paginas favoritas </h1>
  </center>
  <hr>
  <p>estas son mis paginas favoritas:</p>
  <p><a href="http://www.politecnicojic.edu.co"> Politecnico JIC </a>
  <br> <a href="http://www.microsoft.com"> microsoft </a>
  <br> <a href="http://www.yahoo.com"> yahoo! </a>
</body>
</html>
```

Y el resultado será:



IMÁGENES

El uso de imágenes es uno de los factores que ha popularizado tanto la WEB. Incluir imágenes en una presentación web es muy sencillo, solo debe tener en cuenta que las imágenes tienen que tener los formatos GIF, JPEG o PNG.

Nota: *El abuso en la cantidad de imágenes que tenga nuestro sitio puede conducirnos a una sobrecarga que se traduce en una distracción para el navegante, quien tendrá más dificultad en encontrar la información necesaria, y un mayor tiempo de carga de la página lo que puede ser de un efecto nefasto si nuestro visitante no tiene una buena conexión o si es un poco impaciente.*

La etiqueta que utilizaremos para insertar una imagen es (image). Esta etiqueta no posee su cierre correspondiente y en ella hemos de especificar obligatoriamente el paradero de nuestro archivo gráfico mediante el atributo src (source).

La sintaxis queda entonces de la siguiente forma:

```

```

Aparte de este atributo, indispensable obviamente para la visualización de la imagen, la etiqueta nos propone otra serie de atributos de mayor o menor utilidad:

1. **Align**= Permite controlar la alineación de una imagen con respecto a una línea de texto adyacente o a otras imágenes en esa línea. Los valores posibles son los ya conocidos left, right, top, middle y bottom.
2. **Alt**= Dentro de las comillas de este atributo colocaremos una brevísima descripción de la imagen. Esta etiqueta no es indispensable pero presenta varias utilidades. Primeramente, durante el proceso de carga de la página, cuando la imagen no ha sido todavía cargada, el navegador mostrará esta descripción, con lo que el navegante se puede hacer una idea de lo que va en ese lugar. Esto no es tan trivial si tenemos en cuenta que algunos usuarios navegan por la red con una opción del navegador que desactiva el muestreo de imágenes, con lo que tales personas podrán siempre saber de qué se trata el gráfico y eventualmente cambiar a modo con imágenes para visualizarla. Además, determinadas aplicaciones para discapacitados o teléfonos vocales que no muestran imágenes ofrecen la posibilidad de leerlas por lo que nunca está de más pensar en estos colectivos. En general podemos considerar como aconsejable el uso de este atributo salvo para imágenes de poca importancia y absolutamente indispensable si la imagen en cuestión sirve de enlace..

3. **width**= Este atributo es opcional pero es recomendable ponerlo para ayudar al navegador a representar la imagen, significa el ancho de la imagen que vamos a representar. Todos los archivos gráficos poseen unas dimensiones de ancho y alto. Estas dimensiones pueden obtenerse a partir del propio diseñador gráfico o bien haciendo clic con el botón derecho sobre la imagen vista por el navegador para luego elegir propiedades sobre el menú que se despliega. El hecho de explicitar en nuestro código las dimensiones de nuestras imágenes ayuda al navegador a confeccionar la página de la forma que nosotros deseamos antes incluso de que las imágenes hayan sido descargadas. Así, si las dimensiones de las imágenes han sido proporcionadas, durante el proceso de carga, el navegador reservará el espacio correspondiente a cada imagen creando un diagrama correcto. El usuario podrá comenzar a leer tranquilamente el texto sin que este se mueva de un lado a otro cada vez que una imagen se cargue.
4. **height**=Al igual que el atributo **width**, es opcional y recomendable ponerlo, este significa el alto de la imagen.
5. **border**= Definen el tamaño en pixels del cuadro que rodea la imagen.

Nota: Una Imagen, lo mismo que un texto, puede servir de enlace. Vista la estructura de los enlaces podemos muy fácilmente adivinar el tipo de código necesario:

```
<a href="archivo.html"></a>.
```

Ejemplo:

Resultará obvio hacer ahora una página que contenga una imagen varias veces repetida pero con distintos atributos.

- Una de las veces que salga debe mostrarse con su tamaño original y con un borde de 3 pixeles.
- En otra ocasión la imagen aparecerá sin borde, con su misma altura y con una anchura superior a la original
- También mostraremos la imagen sin borde, con su misma anchura y con una altura superior a la original
- Mostraremos la imagen con una altura y anchura mayores que las originales, pero proporcionalmente igual que antes.

Las dimensiones originales de la imagen son 80x80, así que este sería el código fuente:

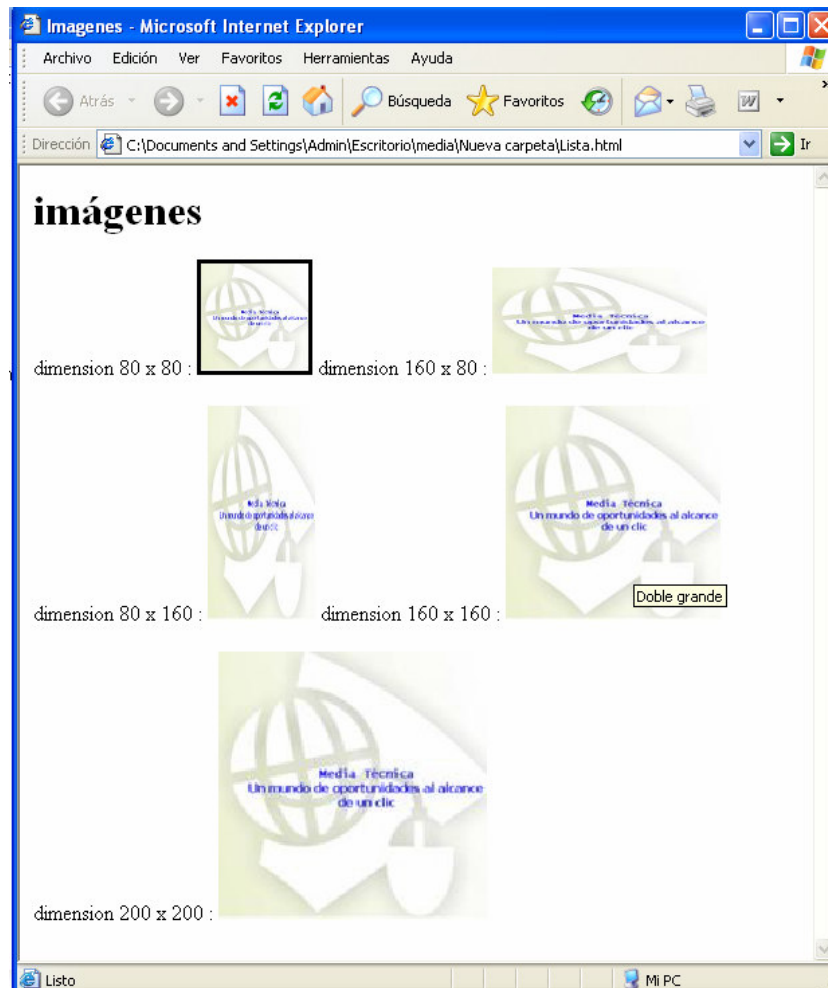
```
<html>
  <head>
    <title> Imagenes </title>
```

```

</head>
<body>
  <h1>imágenes</h1>
  <p> dimension 80 x 80 :
  
  dimension 160 x 80 :
  
  <p> dimension 80 x 160 :
  
  dimension 160 x 160 :
  
  <p> dimension 200 x 200 :
  
</body>
</html>

```

Y el resultado será:



ACTIVIDAD...

- *Desarrollo de las siguientes páginas:*
 - *Una página con la autobiografía del estudiante*
 - *Una página con sus aficiones principales*
 - *Una página que hable de la materia que más le guste en su I.E.*

En todas las páginas deben existir imágenes, enlaces dentro de las mismas páginas, y aplicar todos los conceptos vistos en esta unidad.

ESTRATEGIAS METODOLÓGICAS

- Presentación de la unidad a cargo del profesor.
- Actividades desarrolladas en sala.

RECURSOS

- Humanos: Profesor y alumnos
- Institucionales: Salón de clase, sala de computo
- Materiales: Texto guía

INDICADORES DE EVALUACIÓN

- Evaluación escrita sobre la unidad
- Evaluación de las actividades desarrolladas en sala.

CRITERIOS DE EVALUACIÓN

- ¿Qué es el lenguaje HTML?
- ¿Cuál es la estructura básica de un documento HTML?
- ¿Cuáles son las principales etiquetas para dar formato al texto?
- ¿Qué es un enlace?
- ¿Cuáles son enlaces que se pueden manejar en una página Web?
- ¿Cómo se trabaja con imágenes en una página Web?

UNIDAD 2

MEJORANDO NUESTRA PÁGINA**INTRODUCCIÓN**

Esta unidad busca que el estudiante aplique conceptos más avanzados que le permitan perfeccionar el diseño y desarrollo de una página web. Tablas, formularios, marcos y sonidos son los temas que le permitirán hacerlo.

JUSTIFICACIÓN

Esta unidad busca que el estudiante distinga los conceptos básicos involucrados en la diseño y desarrollo de una página web, cual es la estructura básica de un documento HTML, como dar forma al texto del documento, como trabajar con imágenes y aprender a trabajar con los enlaces (hipervinculos).

OBJETIVO GENERAL

Reconocer otras etiquetas que permitan darle un manejo más uniforme a la página o sitio web

OBJETIVOS ESPECÍFICOS

- Aprender el funcionamiento de las tablas
- Conocer el funcionamiento de los formularios
- Conocer el funcionamiento de los marcos (Frames)
- Aprender a trabajar con el sonido como parte integral de una página

CONTENIDO

1. Tablas
2. Formularios
3. Frames (marcos)
4. Sonido

TABLAS

Las tablas surgieron con la versión HTML 3.0. Las tablas nos permiten representar y ordenar cualquier elemento de nuestra presentación en diferentes filas y columnas de modo que podamos resumir grandes cantidades de información de una manera que puede representarse rápida y fácilmente..

En un principio nos podría parecer que las tablas son raramente útiles y que pueden ser utilizadas principalmente para listar datos como agendas, resultados y otros datos de una forma organizada. Nada más lejos de la realidad.

Hoy, gran parte de los diseñadores de páginas basan su diseño en este tipo de artilugios. En efecto, una tabla nos permite organizar y distribuir los espacios de la manera más optima. Nos puede ayudar a generar texto en columnas como los periódicos, prefijar los tamaños ocupados por distintas secciones de la página o poner de una manera sencilla un pie de foto a una imagen.

Puede que en un principio nos resulte un poco complicado trabajar con estas estructuras pero, si deseamos crear una página de calidad, tarde o temprano tendremos que vérnoslas con ellas y nos daremos cuenta de las posibilidades nos ofrecen.

Para empezar, nada más sencillo que por el principio: las tablas son definidas por las etiquetas `<table>` y `</table>`.

Dentro de estas dos etiquetas colocaremos todas las otras etiquetas, textos e imágenes que darán forma y contenido a la tabla.

Las tablas son descritas por líneas de izquierda a derecha. Cada una de estas líneas es definida por otra etiqueta y su cierre: `<tr>` y `</tr>`

Asimismo, dentro de cada línea, habrá diferentes celdas. Cada una de estas celdas será definida por otro par de etiquetas: `<td>` y `</td>`. Dentro de estas etiquetas será donde coloquemos nuestro contenido.

Ejemplo:

```
<table>
  <tr>
    <td>Celda 1, linea 1</td>
    <td> Celda 2, linea 1</td>
  </tr>
  <tr>
    <td> Celda 1, linea 2</td>
    <td> Celda 2, linea 2</td>
  </tr>
</table>
```

El resultado:

Celda 1, linea 1 Celda 2, linea 1
Celda 1, linea 2 Celda 2, linea 2

Como vimos las celdas son delimitadas por las etiquetas `<td>` o por las etiquetas `<th>` (si queremos texto en negrita y centrado) y constituyen un entorno independiente del resto del documento. Esto quiere decir que:

Podemos usar prácticamente cualquier tipo de etiqueta dentro de la etiqueta `<td>` para, de esta forma, dar forma a su contenido. Las etiquetas situadas en el interior de la celda no modifican el resto del documento. Las etiquetas de fuera de la celda no son tenidas en cuenta por ésta.

Así pues, podemos especificar el formato de nuestras celdas a partir de etiquetas introducidas en su interior o mediante atributos colocados dentro de la etiqueta de celda `<td>` o bien, en algunos casos, dentro de la etiqueta `<tr>`, si deseamos que el atributo sea válido para toda la línea. La forma más útil y actual de dar forma a las celdas es a partir de las hojas de estilo en cascada (ver unidad 3).

Veamos a continuación algunos atributos útiles para la construcción de nuestras tablas. Empezamos viendo atributos que nos permiten modificar una celda en concreto o toda una línea:

Align	Justifica el texto de la celda del mismo modo que si fuese el de un párrafo.
Valign	Podemos elegir si queremos que el texto aparezca arriba (top), en el centro (middle) o abajo (bottom) de la celda.
Bicolor	Da color a la celda o línea elegida.
Bordercolor	Define el color del borde.

Otros atributos que pueden ser únicamente asignados a una celda y no al conjunto de celdas de una línea son:

Background	Nos permite colocar un fondo para la celda a partir de un enlace a una imagen.
Height	Define la altura de la celda en pixels o porcentaje.
Width	Define la anchura de la celda en pixels o porcentaje.
Colspan	Expande una celda horizontalmente.
Rowspan	Expande una celda verticalmente.

Además de los atributos específicos de cada celda o línea, las tablas pueden ser adicionalmente formateadas a partir de los atributos que nos ofrece la propia etiqueta `<table>`. He aquí aquellos que pueden parecernos en un principio importantes:

Align	Alinea horizontalmente la tabla con respecto a su entorno.
Background	Nos permite colocar un fondo para la tabla a partir de un enlace a una imagen.
Bicolor	Da color de fondo a la tabla.
Border	Define el número de pixels del borde principal.
Bordercolor	Define el color del borde.
Cellpadding	Define, en pixels, el espacio entre los bordes de la celda y el contenido de la misma.
Cellspacing	Define el espacio entre los bordes (en pixels).
height	Define la altura de la tabla en pixels o porcentaje.
width	Define la anchura de la tabla en pixels o porcentaje.

Los atributos que definen las dimensiones, height y width, funcionan de una manera análoga a la de las celdas tal y como hemos visto en el capítulo anterior. Contrariamente, el atributo align no nos permite justificar el texto de cada una de las celdas que componen la tabla, sino más bien, justificar la propia tabla con respecto a su entorno.

Tablas anidadas

Muy útil también es el uso de tablas anidadas. De la misma forma que podíamos incluir listas dentro de otras listas, las tablas pueden ser incluidas dentro de otras. Así, podemos incluir una tabla dentro de la celda de otra. El modo de funcionamiento sigue siendo el mismo aunque la situación puede complicarse si el número de tablas embebidas dentro de otras es elevado.

Consejo: Muchas páginas (La mayoría de las páginas avanzadas) que basan su diseño en tablas, realizan anidaciones de tablas constantemente para meter unos elementos de la página dentro de otros. Se pueden anidar tablas sin límite, sin embargo, en el caso de Netscape 4 hay que tener cuidado con el número de tablas que anidamos, porque a medida que metemos una tabla dentro de otra y otra dentro de esta y otra más, aumentando el grado de anidación sucesivamente... podemos encontrar problemas en su visualización y puede que la página tarde un poco de tiempo más en mostrarse en pantalla.

Vamos a ver un código de anidación de tablas. Veamos primero el resultado y luego el código, así conseguiremos entenderlo mejor.

Celda de la tabla principal	<table border="1"><tr><td>Tabla anidada, celda 1</td><td>Tabla anidada, celda 2</td></tr><tr><td>Tabla anidada, celda 3</td><td>Tabla anidada, celda 4</td></tr></table>	Tabla anidada, celda 1	Tabla anidada, celda 2	Tabla anidada, celda 3	Tabla anidada, celda 4
Tabla anidada, celda 1	Tabla anidada, celda 2				
Tabla anidada, celda 3	Tabla anidada, celda 4				

Este sería el código:

```
<table cellspacing="10" cellpadding="10" border="3">
<tr>
  <td align="center">
    Celda de la tabla principal
  </td>
  <td align="center">
    <table cellspacing="2" cellpadding="2" border="1">
      <tr>
        <td>Tabla anidada, celda 1</td>
        <td>Tabla anidada, celda 2</td>
      </tr>
      <tr>
        <td>Tabla anidada, celda 3</td>
        <td>Tabla anidada, celda 4</td>
      </tr>
    </table>
  </td>
</tr>
</table>
```

ACTIVIDAD...

Para practicar un poco lo que acabamos de ver vamos a realizar un ejercicio, en el que simplemente queremos construir una página que contenga las siguientes tablas:

Primer tabla

Animales en peligro de extinción			
Nombre	Cabezas	Previsión 2010	Previsión 2020
Ballena	6000	4000	1500
Oso Pardo	50	0	
Lince	10		
Tigre	300	210	

Segunda tabla

Climas de América del Sur					
Parte de arriba de América del Sur. Países como:	Venezuela	Parte de abajo de América del Sur. Países como:	Argentina		
	Colombia		Chile		
	Ecuador		Uruguay		
	Perú		Paraguay		
Bosque tropical, clima de sabana, clima marítimo con inviernos secos.			Climas marítimos con veranos secos, con inviernos secos, climas fríos, clima de estepa, clima desértico.		

FORMULARIOS

La Web se ha convertido en una poderosa arma para las empresas que se dedican a realizar encuestas y, los formularios han sido una de las herramientas que han ayudado a este auge.

Los formularios nos van a permitir, desde dentro de una presentación web, solicitar información al visitante. Estos formularios estarán compuestos por tantos campos como informaciones queramos obtener. Una vez introducidos los valores en estos campos serán enviados a una URL donde se procesará toda esta información.

Los formularios son esas famosas cajas de texto y botones que podemos encontrar en muchas páginas web. Son muy utilizados para realizar búsquedas o bien para introducir datos personales por ejemplo en sitios de comercio electrónico. Los datos que el usuario introduce en estos campos son enviados al correo electrónico del administrador del formulario o bien a un programa que se encarga de procesarlo automáticamente.

Usando HTML podemos únicamente enviar el formulario a un correo electrónico. Si queremos procesar el formulario mediante un programa la cosa puede resultar un poco más compleja ya que tendremos que emplear otros lenguajes más sofisticados. En este caso, la solución más sencilla es utilizar los programas prediseñados que nos proponen un gran número de servidores de alojamiento y que nos permiten almacenar y procesar los datos en forma de archivos u otros formatos. Si sus páginas están alojadas en un servidor que no propone este tipo de ventajas, siempre se puede recurrir a servidores de terceros que ofrecen este u otro tipo de servicios gratuitos para webs. Por supuesto, existe otra

alternativa que es la de aprender lenguajes como ASP o PHP que nos permitirán, entre otras cosas, el tratamiento de formularios.

Los formularios son definidos por medio de las etiquetas `<form>` y `</form>`. Entre estas dos etiquetas colocaremos todos los campos y botones que componen el formulario. Dentro de esta etiqueta `<form>` debemos especificar algunos atributos:

Action: Define el tipo de acción a llevar a cabo con el formulario. Como ya hemos dicho, existen dos posibilidades:

- El formulario es enviado a una dirección de correo electrónico
- El formulario es enviado a un programa o script que procesa su contenido

En el primer caso, el contenido del formulario es enviado a la dirección de correo electrónico especificada por medio de una sintaxis de este tipo:

```
<form action="mailto:direccion@correo.com" ...>
```

Si lo que queremos es que el formulario sea procesado por un programa, hemos de especificar la dirección del archivo que contiene dicho programa. La etiqueta quedaría en este caso de la siguiente forma:

```
<form action="dirección del archivo" ...>
```

La forma en la que se expresa la localización del archivo que contiene el programa es la misma que la vista para los enlaces.

Method: Este atributo se encarga de especificar la forma en la que el formulario es enviado. Los dos valores posibles que puede tomar esta atributo son *post* y *get*. Para efectos prácticos y, salvo que se diga lo contrario, daremos siempre el valor *post*.

Enctype: Se utiliza para indicar la forma en la que viajará la información que se mande por el formulario. En el caso más corriente, enviar el formulario por correo electrónico, el valor de este atributo debe de ser *text/plain*. Así conseguimos que se envíe el contenido del formulario como texto plano dentro del email.

Si queremos que el formulario se procese automáticamente por un programa, generalmente no utilizaremos este atributo, de modo que tome su valor por defecto, es decir, no incluiremos *enctype* dentro de la etiqueta `<form>`

Ejemplo de etiqueta <form> completa

Así, para el caso más habitual -el envío del formulario por correo- la etiqueta de creación del formulario tendrá el siguiente aspecto:

```
<form action="mailto:direccion@correo.com (o nombre del archivo de proceso)" method="post" enctype="text/plain">
```


Entre esta etiqueta y su cierre colocaremos el resto de etiquetas que darán forma a nuestro formulario, las cuales veremos mas adelante.

Elementos de los Formularios.

El HTML nos propone una gran diversidad de alternativas a la hora de crear nuestros formularios. Estas van desde la clásica caja de texto hasta la lista de opciones pasando por las cajas de validación.

Veamos en qué consiste cada una de estas modalidades y como podemos implementarlas en nuestro formulario.

- **Texto corto:**

Las cajas de texto son colocadas por medio de la etiqueta `<input>`. Dentro de esta etiqueta hemos de especificar el valor de dos atributos: *type* y *name*.

La etiqueta es de la siguiente forma:

```
<input type="text" name="nombre">
```

De este modo expresamos nuestro deseo de crear una caja de texto cuyo contenido será llamado nombre (por ejemplo). El aspecto de este tipo de cajas es:



El nombre del elemento del formulario es de gran importancia para poder identificarlo en nuestro programa de procesamiento o en el mail recibido. Por otra parte, es importante indicar el atributo *type*, ya que, como veremos, existen otras modalidades de formulario que usan esta misma etiqueta.

El empleo de estas cajas esta fundamentalmente destinado a la toma de datos breves: palabras o conjuntos de palabras de longitud relativamente corta. Veremos más adelante que existe otra forma de tomar textos más largos a partir de otra etiqueta.

Además de estos dos atributos, esenciales para el correcto funcionamiento de nuestra etiqueta, existen otra serie de atributos que pueden resultarnos de utilidad pero que no son imprescindibles:

Size: Define el tamaño de la caja en número de caracteres. Si al escribir el usuario llega al final de la caja, el texto ira desfilando a medida que se escribe haciendo desaparecer la parte de texto que queda a la izquierda.

Maxlength: Indica el tamaño máximo del texto que puede ser tomado por el formulario. Es importante no confundirlo con el atributo *size*. Mientras el primero define el tamaño aparente de la caja de texto, *maxlength* indica el tamaño máximo

real del texto que se puede escribir. Podemos tener una caja de texto con un tamaño aparente (*size*) que es menor que el tamaño máximo (*maxlength*). Lo que ocurrirá en este caso es que, al escribir, el texto ira desfilando dentro de la caja hasta que lleguemos a su tamaño máximo definido por *maxlength*, momento en el cual nos será imposible continuar escribiendo.

Value: En algunos casos puede resultarnos interesante asignar un valor definido al campo en cuestión. Esto puede ayudar al usuario a rellenar más rápidamente el formulario o darle alguna idea sobre la naturaleza de datos que se requieren. Este valor inicial del campo puede ser expresado mediante el atributo *value*. Veamos su efecto con un ejemplo sencillo:

```
<input type="text" name="nombre" value="Pepito Perez">
```

Genera un campo de este tipo:



Nota: *Nota: estamos obligados a utilizar la etiqueta <form>*

Aunque de lo que se lee en estos capítulos sobre formularios se puede entender bien esto, hemos querido remarcarlo para que quede muy claro: Cuando queremos utilizar en cualquier situación elementos de formulario debemos escribirlos siempre entre las etiquetas <form> y </form>. De lo contrario, los elementos se verán perfectamente en Explorer pero no en Netscape.

Dicho de otra forma, en Netscape no se visualizan los elementos de formulario a no ser que estén colocados entre las correspondientes etiquetas de inicio y fin de formulario.

Es por ello que para mostrar un campo de texto no vale con poner la etiqueta <input>, sino que habrá que ponerla dentro de un formulario. Así:

```
<form>  
  <input type="text" name="nombre" value="Pepito Perez">  
</form>
```

Veremos posteriormente que este atributo puede resultar relevante en determinadas situaciones.

- **Texto oculto**

Podemos esconder el texto escrito por medio de asteriscos de manera a aportar una cierta confidencialidad. Este tipo de campos son análogos a los de texto con una sola diferencia: reemplazamos el atributo `type="text"` por `type="password"`:

```
<input type="password" name="nombre">
```

En este caso, podemos comprobar que al escribir dentro del campo en lugar de texto vemos asteriscos.

Estos campos son ideales para la introducción de datos confidenciales, principalmente códigos de acceso.

- **Texto largo**

Si deseamos poner a la disposición de usuario un campo de texto donde pueda escribir cómodamente sobre un espacio compuesto de varias líneas, hemos de invocar una nueva etiqueta: `<textarea>` y su cierre correspondiente.

Este tipo de campos son prácticos cuando el contenido a enviar no es un nombre teléfono o cualquier otro dato breve, sino más bien, un comentario, opinión, etc.

Dentro de la etiqueta *textarea* deberemos indicar, como para el caso visto anteriormente, el atributo *name* para asociar el contenido a un nombre que será asemejado a una variable en los programas de proceso. Además, podemos definir las dimensiones del campo a partir de los atributos siguientes:

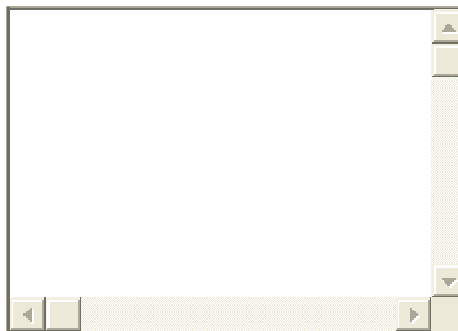
Rows: Define el número de líneas del campo de texto.

Cols: Define el número de columnas del campo de texto.

La etiqueta queda por tanto de esta forma:

```
<textarea name="comentario" rows="10" cols="40"></textarea>
```

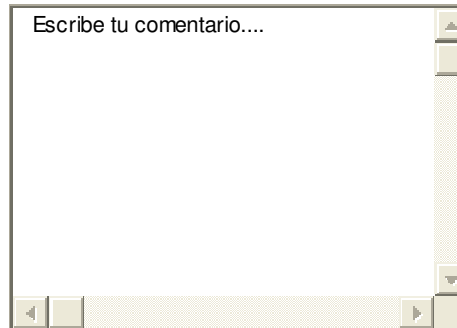
El resultado es el siguiente:



Asimismo, es posible predefinir el contenido del campo. Para ello, no usaremos el atributo *value* sino que escribiremos dentro de la etiqueta el contenido que deseamos atribuirle. Veámoslo:

```
<textarea name="comentario" rows="10" cols="40"> Escribe tu comentario .... </textarea>
```

Dará como resultado:



Los textos son una manera muy práctica de hacernos llegar la información del navegante. No obstante, en muchos casos, los textos son difícilmente adaptables a programas que puedan procesarlos debidamente o bien, puede que su contenido no se ajuste al tipo de información que requerimos. Es por ello que, en determinados casos, puede resultar más efectivo proponer una elección al navegante a partir del planteamiento de una serie de opciones.

Este es el caso de, por ejemplo, ofrecer una lista de países, el tipo de tarjeta de crédito para un pago,...

Este tipo de opciones pueden ser expresadas de diferentes formas. Veamos a continuación cuales son:

- **Listas de opciones**

Las listas de opciones son ese tipo de menús desplegables que nos permiten elegir una (o varias) de las múltiples opciones que nos proponen. Para construirlas emplearemos una etiqueta con su respectivo cierre: `<select>`

Como para los casos ya vistos, dentro de esta etiqueta definiremos su nombre por medio del atributo *name*. Cada opción será incluida en una línea precedida de la etiqueta `<option>`.

Podemos ver, a partir de estas directivas, la forma más típica y sencilla de esta etiqueta:

```
<select name="estacion">
  <option>Primavera</option>
  <option>Verano</option>
```

```
<option>Otoño</option>
<option>Invierno</option>
</select>
```

El resultado es:



Esta estructura puede verse modificada principalmente a partir de otros dos atributos:

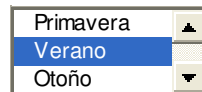
Size: Indica el número de valores mostrados de la lista. El resto pueden ser vistos por medio de la barra lateral de desplazamiento.

Multiple: Permite la selección de más varios elementos de la lista. La elección de más de un elemento se hace como con el explorador de Windows, a partir de las teclas *ctrl* o *shift*. Este atributo se expresa sin valor alguno, es decir, no se utiliza con el igual: simplemente se pone para conseguir el efecto, o no se pone si queremos una lista desplegable común.

Veamos cual es el efecto producido por estos dos atributos cambiando la línea:

```
<select name="estacion"> por: <select name="estacion" size="3" multiple>
```

La lista quedará de esta forma:



La etiqueta `<option>` puede asimismo ser matizada por medio de otros atributos

Selected: Del mismo modo que *multiple*, este atributo no toma ningún valor sino que simplemente indica que la opción que lo presenta esta elegida por defecto.

Value: Define el valor de la opción que será enviado al programa o correo electrónico si el usuario elige esa opción. Este atributo puede resultar muy útil si el formulario es enviado a un programa puesto que a cada opción se le puede asociar un número o letra, lo cual es más fácilmente manipulable que una palabra o texto. Podríamos así escribir líneas del tipo:

```
<option value="1">Primavera</option>
```

De este modo, si el usuario elige primavera, lo que le llegara al programa (o correo) es una variable llamada *estacion* que tendrá como valor 1. En el correo electrónico recibiríamos:

```
estacion=1
```

- **Botones de radio**

Existe otra alternativa para plantear una elección, en este caso, obligamos al internauta a elegir únicamente una de las opciones que se le proponen.

La etiqueta empleada en este caso es `<input>` en la cual tendremos el atributo `type` ha de tomar el valor `radio`. Veamos un ejemplo:

```
<input type="radio" name="estacion" value="1">Primavera  
<br>  
<input type="radio" name="estacion" value="2">Verano  
<br>  
<input type="radio" name="estacion" value="3">Otoño  
<br>  
<input type="radio" name="estacion" value="4">Invierno
```

Hay que fijarse que la etiqueta `<input type="radio">` sólo coloca la casilla pinchable en la página. Los textos que aparecen al lado, así como los saltos de línea los colocamos con el correspondiente texto en el código de la página y las etiquetas HTML que necesitamos.

El resultado es el siguiente:

Primavera
 Verano
 Otoño
 Invierno

Como puede verse, a cada una de las opciones se le atribuye una etiqueta `input` dentro de la cual asignamos el mismo nombre (`name`) para todas las opciones y un valor (`value`) distinto. Si el usuario elige supuestamente *Otoño*, recibiremos en nuestro correo una línea como esta:

```
estacion=3
```

Cabe señalar que es posible preseleccionar por defecto una de las opciones. Esto puede ser conseguido por medio del atributo `checked`:

```
<input type="radio" name="estacion" value="2" checked>Verano
```

Veamos el efecto:

Primavera
 Verano

- Otoño
- Invierno

- **Cajas de validación**

Este tipo de elementos pueden ser activados o desactivados por el visitante por un simple clic sobre la caja en cuestión. La sintaxis utilizada es muy similar a las vistas anteriormente:

```
<input type="checkbox" name="pollo">Me gusta el pollo
```

El efecto:

Me gusta el pollo

La única diferencia fundamental es el valor adoptado por el atributo *type*.

Del mismo modo que para los botones de radio, podemos activar la caja por medio del atributo *checked*.

El tipo de información que llegara a nuestro correo (o al programa) será del tipo:

pollo=on (u off dependiendo si ha sido activada o no)

Los formularios han de dar plaza no solamente a la información a tomar del usuario sino también a otra serie de funciones. Concretamente, han de permitirnos su envío mediante un botón. También puede resultar práctico poder proponer un botón de borrado o bien acompañarlo de datos ocultos que puedan ayudarnos en su procesamiento.

A continuación, para terminar el tema de formularios, daremos a conocer los medios de instalar otras estas funciones.

- **Botón de envío**

Para dar por finalizado el proceso de relleno del formulario y hacerlo llegar a su gestor, el navegante ha de validarlo por medio de un botón previsto a tal efecto. La construcción de dicho botón no reviste ninguna dificultad una vez familiarizados con las etiquetas *input* ya vistas:

```
<input type="submit" value="Enviar">
```

Con este código generamos un botón como este:

Enviar

Como puede verse, tan solo hemos de especificar que se trata de un botón de envío (`type="submit"`) y hemos de definir el mensaje del botón por medio del atributo `value`.

- **Botón de borrado**

Este botón nos permitirá borrar el formulario por completo en el caso de que el usuario desee rehacerlo desde el principio. Su estructura sintáctica es análoga a la anterior:

```
<input type="reset" value="Borrar">
```

A diferencia del botón de envío, indispensable en cualquier formulario, el botón de borrado resulta meramente optativo y no es utilizado frecuentemente. Hay que tener cuidado de no ponerlo muy cerca del botón de envío y de distinguir claramente el uno del otro.

- **Datos ocultos**

En algunos casos, aparte de los propios datos enviados por el usuario, puede resultar práctico enviar datos definidos por nosotros mismos que ayuden al programa en su procesamiento del formulario. Este tipo de datos, que no se muestran en la página pero si pueden ser detectados solicitando el código fuente, no son frecuentemente utilizados por páginas construidas en HTML, son más bien usados por páginas que emplean tecnologías de servidor.

```
<input type=hidden name="sitio" value="www.poltecnicojic.edu.co">
```

Esta etiqueta, incluida dentro de nuestro formulario, enviará un dato adicional al correo o programa encargado de la gestión del formulario. Podríamos, a partir de este dato, dar a conocer al programa el origen del formulario o algún tipo de acción a llevar a cabo (una redirección por ejemplo).

- **Botones normales**

Dentro de los formularios también podemos colocar botones normales, pulsables como cualquier otro botón. Igual que ocurre con los campos `hidden`, estos botones por si solos no tienen mucha utilidad pero podremos necesitarlos para realizar acciones en el futuro. Su sintaxis es la siguiente.

```
<input type=button value="Texto escrito en el botón">
```

El uso más frecuente de un botón es en la programación en el cliente. Utilizando lenguajes como Javascript podemos definir acciones a tomar cuando un visitante pulse el botón de una página web.

Ejemplo de formulario

Pasemos ahora a ejemplificar todo lo aprendido a partir de la creación de un formulario que consulta el grado de satisfacción de los usuarios de una línea de autobuses ficticia. El

formulario está construido para que envíe los datos por correo electrónico a un buzón determinado.

Vemos el formulario en esta página. Vosotros tratar de construirlo para ver si se ha entendido bien los temas sobre formularios.

Nombre

Email

Población

Sexo

Hombre

Mujer

Frecuencia de los viajes

Comentarios sobre su satisfacción personal

Deseo recibir notificación de las novedades en las líneas de autobuses.

El código fuente de este formulario es:

```
<form action = "mailto:jjmr13@gmail.com" method = "post" enctype =
"text/plain" >
Nombre <input type="text" name="nombre" size="30" maxlength="100">
<br>
Email <input type="text" name="email" size="25" maxlength="100"
value="@">
<br>
Población <input type="text" name="poblacion" size="20" maxlength="60">
<br>
Sexo
<br>
<input type="radio" name="sexo" value="Varon" checked> Hombre
<br>
<input type="radio" name="sexo" value="Mujer"> Mujer
<br>
<br>
Frecuencia de los viajes
<br>
<select name="utilizacion">
  <option value="1">Varias veces al dia
  <option value="2">Una vez al dia
  <option value="3">Varias veces a la semana
  <option value="4">varias veces al mes
</select>
<br>
<br>
Comentarios sobre su satisfacción personal
<br>
<textarea cols="30" rows="7" name="comentarios"></textarea>
<br>
<br>
<input type="checkbox" name="recibir_info" checked>
Deseo recibir notificación de las novedades en las líneas de autobuses.
<br>
<br>
<input type="submit" value="Enviar formulario">
<br>
<br>
<input type="Reset" value="Borrar todo">
</form>
```

Para acabar, vamos a ver lo que recibirían por correo electrónico en la empresa de autobuses cuando un usuario cualquiera rellenase este formulario y pulsase sobre el botón de envío.

```
nombre=juan jose restrepo
email= jjr@gmail.com
poblacion=Medellín, Antioquia
```

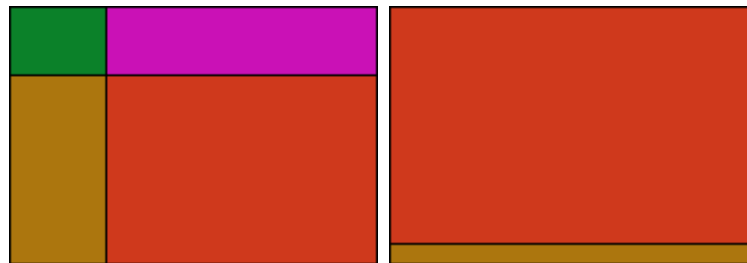
```
sexo=Varon
utilizacion=2
comentarios=No creo que sea una buena línea. Poner más autobuses.
recibir_info=on
```

FRAMES (MARCOS)

Frames (en inglés, marcos o cuadros) es un procedimiento del lenguaje HTML para dividir la pantalla en diferentes zonas, o ventanas, que pueden actuar independientemente unas de otras, como si se trataran de páginas diferentes, pues incluso cada una de ellas pueden tener sus propias barras deslizadoras. Los navegadores que lo implementan son el Netscape 2.0, y el Explorer 2.0 en adelante.

Una de sus características más importantes es que pulsando un enlace situado en un *frame*, se puede cargar en otro *frame* una página determinada. Esto se utiliza frecuentemente para tener un *frame* estrecho en la parte lateral (o superior) con un índice del contenido en forma de diferentes enlaces, que, al ser pulsados cargan en la ventana principal las distintas páginas. De esta manera se facilita la navegación entre las páginas, pues aunque se vaya pasando de unas a otras, siempre estará a la vista el índice del conjunto.

Un ejemplo de las áreas que se pueden construir en una construcción de frames se puede ver en las imágenes siguientes.



Para comprender los distintos conceptos vamos a desarrollar un ejemplo, creando una página con dos *Frames*. El de la izquierda va a servir de índice de lo que veamos en el de la derecha, y en éste veremos inicialmente una página de presentación.

Documento de definición de los *frames*

Lo primero que tenemos que hacer es crear un documento HTML en el que definiremos cuántas zonas va a haber, qué distribución y tamaño van a tener, y cuál va ser el contenido de cada una de ellas.

En el ejemplo que vamos a desarrollar, la página va a tener dos *frames* distribuidos en columnas (es decir, uno al lado del otro, en vez de uno encima del otro, lo que sería una distribución en filas).

Con respecto al tamaño, haremos que el primero (el del izquierda) ocupe el 20% del ancho de la pantalla, y el otro, el 80% restante.

Y con respecto al contenido, el *frame* de la izquierda va a contener un documento HTML que va a servir de índice de lo que veamos en el otro (y que vamos a llamar `mi_indice.html`), y el de la derecha otro documento HTML que va a servir de página de presentación (al que llamaremos `mi_presentacion.html`).

Todo lo anterior se refleja en el siguiente documento HTML:

```
<html>
  <head>
    <title>mi primera pagina con frames</title>
  </head>
  <frameset cols="20%, 80%">
    <frame src= "mi_indice.html">
    <frame src= "mi_presentacion.html" name="principal">
  </frameset>
</html>
```

Obsérvese lo siguiente:

Es un documento parecido a los que conocíamos hasta ahora. La diferencia está en que en vez de utilizar la etiqueta BODY, que sirve normalmente para delimitar lo que se va a ver en la pantalla, se hace uso de la etiqueta FRAMESET (definir los *frames*).

En este caso, con la etiqueta `<FRAMESET COLS="20%, 80%">` se define que va a haber dos *frames* y que van a ir en columnas. Si hubiéramos querido que fueran en filas, habríamos puesto ROWS (filas, en inglés). También se define el espacio en anchura que van a ocupar cada uno de ellos en la pantalla. Se ha puesto como porcentajes del total, pero se podría también haber puesto una cifra absoluta, que representaría el número de pixels a ocupar.

Ya se ha definido el número de *frames*, su distribución y su tamaño, pero falta por definir el contenido de cada *frame*. Esto se hace con las etiquetas:

```
<frame src="mi_indice.html">
<frame src="mi_presentacion.html" name="principal">
```

Con esto se define que el contenido del primer *frame* (el de la izquierda) sea el documento HTML `mi_indice.html` y el del segundo (el de la derecha) sea el documento HTML `mipresentacion.html`.

Obsérvese que en la etiqueta del segundo se ha incluido el atributo NAME="principal", pero no así en el primero. El motivo es que se necesita dar un nombre al segundo *frame*, pues, como veremos a continuación, en el documento del primer *frame* va a haber unos enlaces que van a ir dirigidos hacia él. En este caso sólo tenemos dos *frames*, pero podría haber más, y es necesario distinguirlos unos de otros. Y el primero no necesita nombre, pues no va a haber enlaces en el segundo dirigidos hacia él.

A este documento le vamos a llamar *mi_pagina.html*, pero todavía no lo vamos a guardar, pues falta por añadir algo que veremos más adelante.

Documentos HTML de cada frame

Necesitamos ahora confeccionar el documento HTML de cada uno de los *frames*. Recuérdese que son como páginas independientes, que pueden tener cada una su propio fondo, etc., y todo lo que queramos añadir en ellos y que hemos aprendido hasta ahora.

Documento del frame de la izquierda

Va a tener un fondo amarillo, y va a contener dos enlaces dirigidos al *frame* de la derecha.

```
<html>
  <head>
    <title> indice </title>
  </head>
  <body bgcolor="#ffbb00">
    <p><a href="mi_presentacion.html" target="principal"> presentación
    </a>
    <p><a href="otra_pagina.html" target="principal"> esta es otra
    página </a>
    <p>
  </body>
</html>
```

Dentro de las etiquetas de los enlaces podemos observar algo nuevo, y es el atributo TARGET (en inglés: objetivo, blanco), que sirve para hacer que al ser activado el enlace no se cargue en el propio frame, sino en otro, precisamente en el que hayamos llamado con ese nombre en el documento de definición de los frames.

En nuestro caso, le hemos dado el nombre de "principal" al *frame* de la derecha, y es por tanto ahí donde se van a cargar los documentos HTML.

Guardamos este documento con el nombre de *mi_indice.html*.

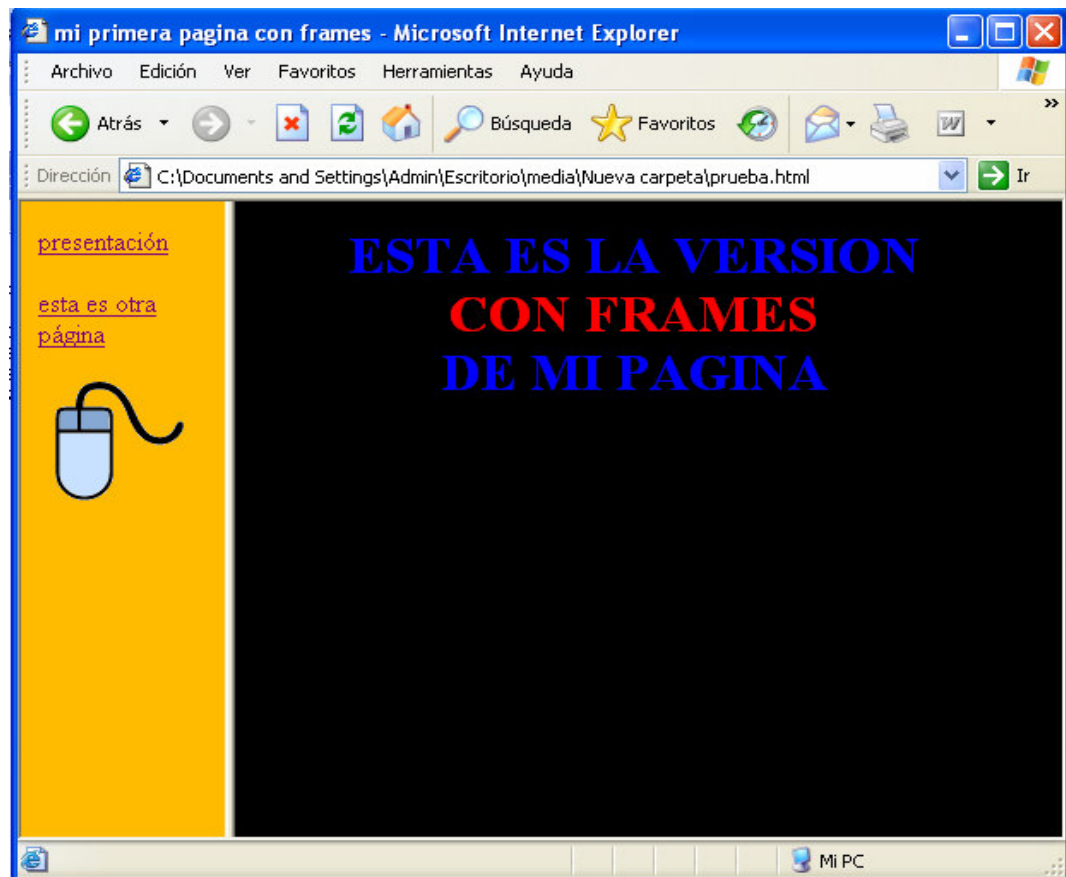
Documento del frame de la derecha

Va a tener un fondo negro, y va a contener solamente un texto.

```
<html>
  <head>
    <title> presentacion </title>
  </head>
  <body bgcolor="#000000" text="#0000ff">
    <center>
      <font size=+3><strong>
        <p>esta es la version
        <br><font color="#ff0000">con frames</font>
        <br>de mi pagina
      </strong>
    </font>
    </center>
  </body>
</html>
```

Guardamos este documento con el nombre de mi_presentacion.html

Y el resultado será:



Alternativa para los navegadores que no soportan frames

Con lo visto anteriormente, ya tenemos las tres piezas que necesitamos:

- Documento de definición de los frames
- Documento del contenido del primer frame
- Documento del contenido del segundo frame

Hay que prever el caso en que los navegadores no soportan o desconocen las etiquetas FRAMESET y FRAME.

Para estos casos está prevista la etiqueta <NOFRAMES> y </NOFRAMES>. Se añaden al final del documento de definición de los *frames*, y a se pone entre ambas lo que queremos que vean los que acceden con un navegador que no soporta *frames*. Puede incluso ser el código HTML de una página completa (lo que normalmente va entre las etiquetas <BODY> y </BODY>)

En nuestro caso, pudiéramos poner un mensaje advirtiendo de esta circunstancia, y dirigiendo al usuario, con un enlace normal, hacia una página que hayamos definido como principal o de presentación para estos casos. (También podríamos no decir nada sobre *frames*, y colocar aquí el código de la página de presentación)

```
<noframes>
  estas utilizando un navegador que no soporta frames. <p>pulsa para visitar
  mi <a href="presentacion.html"> página</a>.
</noframes>

<html>
  <head>
    <title>mi pagina con frames</title>
  </head>
  <frameset cols="20%, 80%">
    <frame src="mi_indice.html">
    <frame src="mi_presentacion.html" name="principal">
  </frameset>
  <noframes>
    estas utilizando un navegador que no soporta frames.
    <p>pulsa para visitar mi <a href="presentacion.html"> página</a>.
  </noframes>
</html>
```

Ahora sí podemos guardar este documento con el nombre de mi_pagina.html.

Atributos de la etiqueta <FRAMESET>

La etiqueta FRAMESET, como hemos visto en el ejemplo, es la que define la distribución, el número y tamaño de los frames. Tiene dos atributos: COLS (columnas) y ROWS (filas):

```
<frameset cols="xx, yy, zz, ..">
<frameset rows="xx, yy, zz, ..">
```

Define la distribución de los *frames* en columnas o en filas, según se use uno u otro atributo.

Define el número de frames que habrá, pues, por ejemplo:

```
<frameset cols="xx, yy"> (habrá dos frames en columnas)
<frameset cols="xx, yy, zz"> (habrá tres frames en columnas)
```

Define el tamaño de los frames, según el valor que demos a XX, YY, ZZ.... Este valor se puede expresar en:

Un porcentaje del ancho del pantalla (para las columnas), o del alto de la pantalla (para las filas). Así, por ejemplo:

```
<FRAMESET COLS="%20, %80"> (la columna de la izquierda ocupará el 20% del ancho de la pantalla, y la de la derecha el 80% restante)
```

```
<FRAMESET ROWS="%10, %70, %20"> (la fila superior ocupará el 10% del alto de la pantalla, la del medio el 70%, y la inferior el 20%)
```

Un número absoluto que representa el número de pixels que ocupará cada *frame* a lo ancho o a lo alto (según sean filas o columnas). Así, por ejemplo:

```
<FRAMESET COLS="40, 600"> (la columna de la izquierda tendrá 40 pixels de ancho y la de la derecha 600). Pero es peligroso utilizar sólo valores absolutos, pues el tamaño de la pantalla varía de un usuario a otro. Si se va a usar algún valor absoluto para un frame, es mejor mezclarlo con alguno relativo, como los que vamos a ver a continuación, para que se ajuste el total a la pantalla del usuario
```

Un valor relativo que se consigue poniendo un asterisco (*), en vez de un número. Esto se interpreta como que ese *frame* debe tener el espacio restante. Por ejemplo:

```
<FRAMESET ROWS="100, *, 100"> (Habrá tres filas, la superior y la inferior de una altura fija de 100 pixels, y la del medio obtendrá el espacio restante).
```

Si hay más de un frame con asterisco, ese espacio restante se dividirá por igual entre ellos. Si hay un número antes del asterisco, ese frame obtiene esa cantidad más de espacio relativo. Así "2*,*" daría dos tercios para el primer frame y un tercio para el otro.

Frames sin bordes

Si se desea que no haya un borde de separación entre los frames, se deben incluir el atributo `FRAMEBORDER=0` dentro de la etiqueta `FRAMESET`. No todas las versiones de los navegadores lo implementan.

Para que también desaparezcan los huecos de separación entre frames hay que añadir otros dos atributos (el primero es para el Explorer y el segundo para el Netscape): FRAMESPACING=0 y BORDER=0. con lo que la etiqueta completa quedaría:

```
<frameset frameborder=0 framespacing=0 border=0 cols="xx, yy">
```

Atributos de la etiqueta <FRAME>

Esta etiqueta define las características de un frame concreto, no del conjunto de los frames, como era el caso con la etiqueta <FRAMESET>. Puede tener los siguientes posibles atributos, que van dentro de la etiqueta <FRAME>:

- **SRC="dirección"**: Esta dirección puede ser la de un documento HTML (tal como hemos utilizado en el ejemplo), o cualquier otro recurso del Web (o URL). Con este atributo se indica lo que se cargará inicialmente en el frame. Si no se le pone este atributo a la etiqueta <FRAME>, entonces dicho frame aparecerá inicialmente vacío, aunque tendrá las dimensiones asignadas.
- **NAME="nombre_de_la_ventana"**: Este atributo se usa para asignar un nombre a un frame. De esta manera se podrá "dar en el blanco" (en inglés, target) en esta página, desde un enlace situado en otra página. Es decir, que pulsando en otra página un enlace, se cargará precisamente en ésta, tal como hemos visto en el ejemplo. El atributo NAME es opcional. Por defecto, todas las ventanas carecen de nombre. Los nombres que se escojan deben comenzar por un carácter alfanumérico (una letra o un número, pero no otro tipo de símbolo).
- **MARGINWIDTH="número"**: Se utiliza este atributo cuando se quiere controlar el ancho de los márgenes dentro de un frame. El número que se ponga representa los pixels de los márgenes. Este atributo es opcional.
- **MARGINHEIGHT="número"**: Igual que el anterior, pero referido a los márgenes en altura.
- **SCROLLING="yes|no|auto"**: Este atributo se utiliza para decidir si el frame tendrá o no una barra deslizador. Si se escoge "yes" tendrá siempre una barra deslizador. Si se escoge "no" no la tendrá nunca, y si se escoge "auto", será el navegador quien decida si la tendrá o no. Este atributo es opcional. Su valor por defecto es "auto".
- **NORESIZE**: A este atributo no se le asigna un valor numérico, como a los demás. Es un indicador para que la ventana no se pueda re-dimensionar (en inglés, resize) por parte del usuario. Es un atributo opcional. Por defecto, todos los frames son re-dimensionables.
- **FRAMEBORDER="no"**: Este atributo elimina el borde en un frame, pero si se quiere que se elimine completamente, también hay que ponérselo al frame contiguo. Si se

quiere eliminar los bordes de todos los frames, se debe colocar en la etiqueta FRAMESET, como hemos visto anteriormente.

- **TARGET:** En el ejemplo hemos visto que, como queríamos que los enlaces situados en el frame de la izquierda surtieran efecto no en él mismo, sino en otro frame, teníamos que poner dentro de cada enlace el atributo TARGET="principal", siendo "principal" el nombre que habíamos dado al segundo frame, en el documento de definición de frames. Es decir, hemos utilizado este atributo de esta manera:

TARGET="nombre_dado_a_otro_frame".

Estos nombres, que los escogemos nosotros, pueden ser cualquiera, pero con la condición que el primer carácter sea alfanumérico (letra o número).

Pero hay unos nombres reservados (es decir, que no se pueden usar para denominar a un frame), que hacen que este atributo efectúe unas funciones especiales. Para que cumplan su cometido, es imprescindible escribir estas palabras reservadas (blank, self y top) en minúsculas.

TARGET="_blank". Hace que se abra una nueva copia del navegador, y el enlace activado se carga en ella, a pantalla completa. Es decir, tendríamos dos copias del navegador (Netscape, Explorer, etc.) funcionando a la vez.

TARGET="_self". Hace que el enlace se cargue en el propio frame.

TARGET="_top". Hace que el enlace se cargue a pantalla completa, suprimiendo todos los frames, pero sin que se cargue una nueva copia del navegador. Este es particularmente útil. Un error muy común es olvidarse de poner este atributo en los enlaces que están en un frame, con lo que resulta que al ser activados, la página llamada se carga dentro del propio frame, lo cual es muy molesto si esa página pertenece a otro sitio del WEB, y aún más grave si esa página tiene a su vez frames. Este inconveniente se evita poniendo este atributo dentro de las etiquetas de los enlaces.

Frames anidados dentro de otros frames

Hasta ahora hemos contemplado sólo la posibilidad de tener una distribución de los frames bien en filas o bien en columnas, (dependiendo que se utilice el atributo ROWS o COLS en la etiqueta FRAMESET), pero no ambos a la vez.

Se pueden obtener distribuciones más complejas anidando filas dentro de una columna, o a la inversa, columnas dentro de una fila.

Supongamos que queremos la siguiente distribución:

Un frame estrecho en horizontal en la parte superior, de lado a lado de la pantalla (altura, el 15%)

Otro frame estrecho en vertical en la parte izquierda, debajo del anterior (anchura, el 20%)

Un tercero ocupando el resto de la pantalla.

Vemos que, en realidad, esto equivale a la siguiente distribución:

Dos filas. La superior ocupa el 15% y la inferior el resto.

La fila inferior está a su vez dividida en dos columnas. La primera (la de la izquierda) ocupa el 20% y la otra, el resto.

El documento de definición de las dos filas (olvidémonos de momento que la de abajo está subdividida), sería:

```
<html>
  <head>
    <title>pagina con dos filas</title>
  </head>
  <frameset rows="15%, *">
    <frame src="documento_fila_superior">
    <frame src="documento_fila_inferior">
  </frameset>
</html>
```

Como la fila inferior, en realidad, son dos columnas (con una distribución del 20% y resto), sustituimos (anidando) la etiqueta <FRAME SRC="documento_fila_inferior"> por:

```
<frameset cols="20%, *">
  <frame src="documento_columna_izqda">
  <frame src="documento_columna_dcha">
</frameset>
```

Con lo que queda el documento definitivo así:

```
<html>
  <head>
    <title>pagina con fila superior y dos columnas inferiores</title>
  </head>
  <frameset rows="15%, *">
    <frame src="documento_fila_superior">
    <frameset cols="20%, *">
      <frame src="documento_columna_izqda">
      <frame src="documento_columna_dcha">
    </frameset>
  </frameset>
</html>
```

Ventajas e inconvenientes del uso de frames

El diseño con frames es un asunto bastante controvertido, ya que distintos diseñadores tendrán unas u otras opiniones.

Ventajas de usar frames

- La navegación de la página será más rápida. Aunque la primera carga de la página sería igual, en sucesivas impresiones de páginas ya tendremos algunos marcos guardados, que no tendrían que volverse a descargar.
- Crear páginas del sitio sería más rápido. Como no tenemos que incluir partes de código como la barra de navegación, título, etc. crear nuevas páginas sería un proceso mucho más rápido.
- Partes de la página (como la barra de navegación) se mantienen fijas y eso puede ser bueno, para que el usuario no las pierda nunca de vista.
- Estas mismas partes visibles constantemente, si contienen enlaces, pueden servir muy bien para mejorar la navegación por el sitio.
- Mantienen una identidad del sitio donde se navega, pues los elementos fijos conservan la imagen siempre visible.

Inconvenientes de usar frames

- QUITAN espacio en la pantalla. El espacio ocupado por los frames fijos se pierde a la hora de hacer páginas nuevas, porque ya está utilizado. En definiciones de pantalla pequeña o dispositivos como Palms, este problema se hace más patente.
- Fuerzan al visitante a entrar por la declaración de frames. Si no lo hacen así, sólo se vería una página interior sin los recuadros. Estos recuadros podrían ser insuficientes para una buena navegación por los contenidos y podrían no conservar una buena imagen corporativa.
- La promoción de la página sería, en principio, más limitada. Esto es debido a que sólo se debería promocionar la portada, pues si se promocionan páginas interiores, podría darse en caso de que los visitantes entrasen por ellas en lugar de por la portada, creándose el problema descrito en el punto anterior.
- A mucha gente le disgustan pues no se sienten libres en la navegación, pues entienden que esas partes fijas están limitando su movilidad por la web. Este efecto se hace más patente si la página con frames tiene enlaces a otras páginas web fuera del sitio y, al pulsar un enlace, se muestra la página nueva con los marcos de la página que tiene frames.

- Algunos navegadores no los soportan. Esto no es muy habitual, pero si estamos haciendo una página que queramos que sea totalmente accesible deberíamos considerarlo importante.
- Los bookmarks o favoritos no funcionan correctamente en muchos casos. Si queremos incluir un favorito a una página de un frame que no sea la portada podemos encontrar problemas.
- Puede que el botón de atrás del navegador no se comporte como deseamos.
- Si quieres actualizar más de un frame con la pulsación de un enlace deberás utilizar Javascript. Además los scripts se pueden complicar bastante cuando se tienen que comunicar varios frames entre si.

Conclusión

El trabajo con frames puede ser más bueno o más malo dependiendo de las características de la página a desarrollar, es tu tarea saber si en tu caso debes utilizarlos o no.

SONIDOS

Una página del WEB puede tener sonidos incorporados, bien sea como un fondo sonoro que se ejecuta automáticamente al cargar la página, o como una opción para que la active el propio usuario.

Capacidades sonoras de los navegadores

Para poder escuchar los sonidos es necesario disponer, como es lógico, de una tarjeta de sonido con sus correspondientes altavoces. Pero esto no es suficiente, pues no todos los programas navegadores están capacitados en la misma medida.

- **Explorer de Microsoft**

Es el que está mejor adaptado para el sonido, pues a partir de la versión 2.0 es capaz de reproducir fondos sonoros sin necesidad de añadir nada, y no hay ninguna complicación con los servidores, como ocurre con el Netscape. Además, a partir de la versión 3.0 del Explorer, es incluso compatible con los plug-ins del Netscape.

- **Netscape**

Las versiones anteriores a la 2.0 no son capaces de reproducir fondos sonoros que se ejecuten automáticamente, sino que requerirá que se activen los programas auxiliares asociados a los formatos, .WAV o .MID.

La versión 2.0 sí es capaz de reproducir un fondo sonoro, pero es necesario que tenga instalado un plug-in llamado Crescendo.

La versión 3.0 lleva implícito el plug-in Live Audio (pero sólo la versión completa, no la reducida). En caso afirmativo, es capaz de reproducir un fondo sonoro.

Pero a todas estas complicaciones de las distintas versiones de los navegadores de los usuarios, hay otra más y es que el servidor donde esté alojada la página del WEB debe tener configurados como MIME los formatos, .MID y .WAV. Si esto no es así, aunque depositemos en el servidor nuestro documento HTML acompañado por el correspondiente archivo de sonido, éste no se ejecutará. En dicho caso, es necesario ponerse en contacto con los administradores del servidor para que configuren como MIME los formatos, .MID y .WAV.

Este problema no existe, sin embargo para el Explorer, con el que no nos tenemos que preocupar por este tema.

Fondo sonoro para el Microsoft Internet Explorer

Para las versiones 2.0 en adelante, se utiliza la etiqueta:

```
<bgsound src="archivo_de_sonido" loop=n>
```

El archivo de sonido puede estar en formato .MID o .WAV.

El atributo LOOP (en inglés, lazo) sirve para especificar el número (n) de veces que se debe ejecutar el archivo de sonido. Si se escoge el número n=-1 o se pone LOOP=INFINITE, el sonido se ejecutará indefinidamente. Se puede omitir este atributo, y entonces el archivo se ejecutará una sola vez.

La etiqueta para que se ejecute el archivo sonido.mid dos veces en el Explorer es:

```
<bgsound src="sonido.mid" loop=2>
```

(Para poder oirlo, hay que estar utilizando el Explorer)

Fondo sonoro para el Netscape

La etiqueta básica para el Netscape es:

```
<embed src="archivo_de_sonido" width=xxx height=yy>
```

Donde WIDTH es la anchura y HEIGHT la altura de una consola que aparece, y que tiene diferentes teclas (Play, Stop, Pausa, etc.). Respecto a las dimensiones XXX e YY estas dependerán del plug-ins que existe para Netscape.

El archivo de sonido puede estar en formato, .MID o .WAV, pero recuérdese la advertencia hecha anteriormente, de que estos formatos deben estar configurados como MIME por el servidor donde esté alojada la página.

Dentro de la etiqueta se pueden añadir los siguientes atributos opcionales:

AUTOSTART="TRUE"	Arranca automáticamente.
LOOP="TRUE"	Se ejecuta ininterrumpidamente

Para conseguir que la consola sea invisible hay que añadirle el atributo HIDDEN="TRUE".

```
<embed src="sonido.mid" hidden="true">
```

Fondo sonoro combinado para el Explorer y el Netscape

Se pueden combinar los dos tipos distintos de etiquetas para conseguir que un fondo sonoro sea escuchado por usuarios que utilicen tanto el Explorer como el Netscape (siempre que éste último esté preparado para ello). En este caso, el Explorer ignorará la etiqueta del Netscape, y a la inversa.

Las dos etiquetas necesarias para que se reproduzca un archivo de sonido ya sea en formato, .WAV o .MID, como sonido de fondo, tanto por el Explorer como por el Netscape (lo hará una sola vez):

```
<bgsound src="sonido.mid"> para el explorer.  
<embed src="sonido.mid" hidden="true"> para el netscape.
```

Como se vio anteriormente para que un archivo de sonido, .MID o .WAV se ejecute como sonido de fondo, tanto por el Explorer como por el Netscape y que se ejecute indefinidamente, a la etiqueta del Explorer basta con añadirle el atributo LOOP=INFINITE, como se ha visto anteriormente. Y con respecto a la etiqueta del Netscape, teóricamente debería también bastar añadirle el atributo LOOP="TRUE", pero curiosamente esto no es así, porque hay que poner las dimensiones de la consola (que no se va a ver).

Las dos etiquetas quedarían de esta manera:

```
<bgsound src="sonido.mid" loop=infinite>  
<embed src="sonido.mid" width=200 height=55 autostart="true" loop="true"  
hidden="true">
```

Activación del sonido por el propio usuario

Se ha visto cómo poner un sonido de fondo en una página. Hay otra opción, mucho más sencilla, y es la de poner un enlace a un archivo de sonido, de tal manera, que al pulsarlo se ejecute el archivo. (Ver la Unidad 1, en la que se explica cómo crear enlaces).

Por ejemplo, poner un enlace a un archivo sonido.mid:

```
Escucha esta <a href= "sonido.mid">musica</a>
```

Al pulsar el enlace se activa, en una ventana aparte, el programa que ejecuta el sonido. Esto es válido para todos los navegadores, incluso las versiones más antiguas, con la única condición de que se haya configurado un programa auxiliar capaz de ejecutar archivos .MID o .WAV.

Se puede hacer el enlace con un icono (Ver el unidad 1).

Por ejemplo al hacer un enlace con un icono llamado snd.gif al archivo de sonido sonido.wav, el enlace quedaría así:

```
<a href= "sonido.wav"><img src= "snd.gif"></a>
```

En el tema anterior creamos una versión con frames, en la que aparece inicialmente en el frame de la derecha una página de presentación.

Se puede colocar a esta página una música de fondo, para que sea ejecutada por el Explorer y el Netscape. Para ello debemos hacer lo siguiente:

En el documento de la derecha que llamamos como presentacion.html debemos añadir entre las etiquetas </SRONG> y (situadas al final), lo siguiente:

```
<br><font color="#ff0000">¡escucha la música de fondo!</font>  
<bgsound src="sonido.mid">  
<embed src="sonido.mid" width=2 height=0 autostart= "true">
```

ACTIVIDAD...

- *Desarrollo de las siguientes páginas:*
 - *Una página principal con enlace a las páginas interiores (indice)*
 - *Una página con un tema de su agrado*
 - *Una página que utilice un formulario para el envío de comentarios*
 - *Una página con marcos que integre a las anteriores*

En todas las páginas deben existir imágenes organizadas por medio de tablas, enlaces dentro de las mismas páginas, y aplicar todos los conceptos vistos en esta unidad.

ESTRATEGIAS METODOLÓGICAS

- Presentación de la unidad a cargo del profesor.
- Actividades desarrolladas en sala.

RECURSOS

- Humanos: Profesor y alumnos
- Institucionales: Salón de clase, sala de computo
- Materiales: Texto guía

INDICADORES DE EVALUACIÓN

- Evaluación escrita sobre la unidad
- Evaluación de las actividades desarrolladas en sala.

CRITERIOS DE EVALUACIÓN

- ¿Qué es una tabla en el lenguaje HTML?
- ¿Qué utilidad tienen las tablas en HTML?
- ¿Cuál es la funcionalidad de un formulario?
- ¿En qué casos se pueden utilizar los frames?
- ¿cómo se involucra el sonido a una página HTML?

UNIDAD 3

PÁGINAS DE ESTILO EN CASCADA Y JAVASCRIPT**INTRODUCCIÓN**

Esta unidad busca que el estudiante conozca las páginas de estilo en cascada (CSS) y la forma de utilizarlas. Además, se busca hacer un primer encuentro con el lenguaje Javascript.

JUSTIFICACIÓN

El lenguaje HTML está limitado a la hora de aplicarle forma a un documento. Esto es así porque fué concebido para otros usos (científicos sobretudo), distinto a los actuales, mucho más amplios.

En estas páginas de CSS se pretende dar a conocer la tecnología con un enfoque práctico para que en poco tiempo se pueda usar las CSS de una manera depurada. No se pretende explorar todos los aspectos de la tecnología ya que para realizar esto necesitaríamos un la extensión de un libro entero.

Javascript es un lenguaje de programación utilizado para crear pequeños programitas encargados de realizar acciones dentro del ámbito de una página web. Con Javascript se pueden crear efectos especiales en las páginas y definir interactividades con el usuario. El navegador del cliente es el encargado de interpretar las instrucciones Javascript y ejecutarlas para realizar estos efectos e interactividades, de modo que el mayor recurso, y tal vez el único, con que cuenta este lenguaje es el propio navegador.

Javascript es el siguiente paso, después del HTML, que puede dar un programador de la web que decida mejorar sus páginas y la potencia de sus proyectos. Es un lenguaje de programación bastante sencillo y pensado para hacer las cosas con rapidez, a veces con ligereza. Incluso las personas que no tengan una experiencia previa en la programación podrán aprender este lenguaje con facilidad y utilizarlo en toda su potencia con sólo un poco de práctica

OBJETIVO GENERAL

Utilizar las paginas de estilo en cascada y el lenguaje JavaScript para perfeccionar las páginas Web

OBJETIVOS ESPECÍFICOS

- Conocer el funcionamiento de las páginas de estilo en cascada (CSS)
- Conocer las instrucciones básicas del lenguaje Javascript

CONTENIDO

1. Introducción a las CSS
2. Características y ventajas de las CSS
3. Usos de las CCS
4. Reglas de importancia en los estilos
5. Sintaxis CSS
6. Atributos de las hojas de estilo
7. Trucos avanzados con CSS
8. Introducción a Javascript
9. Algo de historia sobre Javascript
10. Diferencias entre Java y Javascript
11. Usos de Javascript
12. Versiones de navegadores y de Javascript
13. Efectos rápidos
14. El lenguaje Javascript
15. Sintaxis Javascript

INTRODUCCIÓN A LAS CSS

El lenguaje HTML está limitado a la hora de aplicarle forma a un documento. Esto es así porque fue concebido para otros usos (científicos sobretodo), distinto a los actuales, mucho más amplios.

Para solucionar estos problemas los diseñadores han utilizado técnicas tales como la utilización de tablas imágenes transparentes para ajustarlas, utilización de etiquetas que no son estándares del HTML y otras. Estas "trampas" han causado a menudo problemas en las páginas a la hora de su visualización en distintas plataformas.

Además, los diseñadores se han visto frustrados por la dificultad con la que, aun utilizando estos trucos, se encontraban a la hora de planear las páginas, ya que muchos de ellos venían planeando páginas sobre el papel, donde el control sobre la forma del documento es absoluto.

Finalmente, otro antecedente que ha hecho necesario el desarrollo de esta tecnología consiste en que las páginas web tienen mezclado en su código HTML el contenido del documento con las etiquetas necesarias para darle forma. Esto tiene sus inconvenientes ya que la lectura del código HTML se hace pesada y difícil a la hora de buscar errores o depurar las páginas. Aunque, desde el punto de vista de la riqueza de la información y la utilidad de las páginas a la hora de almacenar su contenido, es un gran problema que estos textos estén mezclados con etiquetas incrustadas para dar forma a estos: se degrada su utilidad.

En estas páginas de CSS pretendemos dar a conocer la tecnología con un enfoque práctico para que en pocos capítulos podáis usar las CSS de una manera depurada, reflejando toda nuestra experiencia en su uso. No se pretende explorar todos los aspectos de la tecnología ya que para realizar esto se necesita la extensión de un libro entero.

CARACTERÍSTICAS Y VENTAJAS DE LAS CSS

El modo de funcionamiento de las CSS consiste en definir, mediante una sintaxis especial, la forma de presentación que le aplicaremos a:

- Un web entero, de modo que se puede definir la forma de todo el web de una sola vez.
- Un documento HTML o página, se puede definir la forma, en un pequeño trozo de código en la cabecera, a toda la página.
- Una porción del documento, aplicando estilos visibles en un trozo de la página.

- Una etiqueta en concreto, llegando incluso a poder definir varios estilos diferentes para una sola etiqueta. Esto es muy importante ya que ofrece potencia en la programación. Se puede definir, por ejemplo, varios tipos de párrafos: en rojo, en azul, con márgenes, sin ellos, entre otros.

La potencia de la tecnología salta a la vista. Pero no solo se queda aquí, ya que además esta sintaxis CSS permite aplicar al documento formato de modo mucho más exacto. Si antes el HTML se quedaba corto para planear las páginas y se tenían que utilizar trucos para conseguir efectos, ahora se tienen muchas más herramientas que permiten definir esta forma:

- Podemos definir la distancia entre líneas del documento.
- Se puede aplicar tabulado a las primeras líneas del párrafo.
- Se pueden colocar elementos en la página con mayor precisión, y sin lugar a errores.
- Y mucho más, como definir la visibilidad de los elementos, márgenes, subrayados, tachados...

Y existen mas ventajas, ya que si con el HTML tan sólo se podían definir atributos en las páginas con píxeles y porcentajes, ahora se pueden definir utilizando muchas más unidades como:

- Pixels (px) y porcentaje (%), como antes.
- Pulgadas (in)
- Puntos (pt)
- Centímetros (cm)

USOS DE LAS CSS

Pequeñas partes de la página

Para definir estilos en secciones reducidas de una página se utiliza la etiqueta ****. Con su atributo **style** indicamos en sintaxis CSS las características de estilos. Se ve con un ejemplo, se pondrá un párrafo en el que determinadas palabras se visualizan en color verde.

```
<p>  
Esto es un párrafo en varias palabras <span style="color:green">en color  
verde</span>. resulta muy fácil.  
</p>
```

Que tiene como resultado:

*Esto es un párrafo con varias palabras **en color verde**. resulta muy fácil.*

Estilo definido para una etiqueta

De este modo se puede hacer que toda una etiqueta muestre un estilo determinado. Por ejemplo, se define un párrafo entero en color rojo y otro en color azul. Para ello se utiliza el atributo **style**, que es admitido por todas las etiquetas del HTML (siempre y cuando se disponga de un navegador compatible con CSS).

```
<p style="color:#990000">  
Esto es un párrafo de color rojo.  
</p>  
<p style="color:#000099">  
Esto es un párrafo de color azul.  
</p>
```

Que tiene como resultado:

Esto es un párrafo de color rojo.
Esto es un párrafo de color azul.

Estilo definido en una parte de la página

Con la etiqueta **<DIV>** se pueden definir secciones de una página y aplicarle estilos con el atributo **style**, es decir, se pueden definir estilos de una vez a todo un bloque de la página.

```
<div style="color:#000099; font-weight:bold">  
<h3>Estas etiquetas van en <i>azul y negrita</i></h3>  
<p>  
Seguimos dentro del DIV, luego permanecen los estilos  
</p>  
</div>
```

Que tiene como resultado:

Estas etiquetas van en azul y negrita
Seguimos dentro del DIV, luego permanecen los estilos

Estilo definido para toda una página

Se pueden definir, en la cabecera del documento, estilos para que sean aplicados a toda la página. Es una manera muy cómoda de darle forma al documento y muy potente, ya que estos estilos serán seguidos en toda la página y se ahorran así muchas etiquetas HTML que apliquen forma al documento. Además, si se desean cambiar los estilos de la página se hace de una sola vez.

Este ejemplo es más complicado, puesto que se utiliza la sintaxis CSS de manera más avanzada.

```
<p style="color:#990000">
Esto es un párrafo de color rojo.
</p>
<p style="color:#000099">
Esto es un párrafo de color azul.
</p>
<div style="color:#000099; font-weight:bold">
<h3>Estas etiquetas van en <i>azul y negrita</i></h3>
<p>
Seguimos dentro del DIV, luego permanecen los estilos
</p>
</div>
```

En el ejemplo se ve que se utiliza la etiqueta <STYLE> colocada en la cabecera de la página para definir los distintos estilos del documento.

A grandes rasgos, entre de <STYLE> y </STYLE>, se coloca el nombre de la etiqueta que queremos definir los estilos y entre llaves -{}- colocamos en sintaxis CSS las características de estilos.

```
<html>
<head>
<title>Ejemplo de estilos para toda una página</title>
<STYLE type="text/css">
<!--
H1 {text-decoration: underline; text-align:center}
P {font-Family:arial,verdana; color: white; background-color: black}
BODY {color:black;background-color: #cccccc; text-indent:1cm}
// -->
</STYLE>
</head>
<body>
<h1>Página con estilos</h1>
Bienvenidos...
<p>Siento ser tan hortera, pero esto es un ejemplo sin mucha
importancia</p>
</body>
</html>
```

Como se puede apreciar en el código, se ha definido que la etiqueta <H1> se presentará

- Subrayado
- Centrada

También, por ejemplo, se ha definido que al cuerpo entero de la página (etiqueta BODY) se le apliquen los estilos siguientes:

- Color del texto negro
- Color del fondo grisáceo
- Margen lateral de 1 centímetro

Caber destacar que si se aplican estilos a la etiqueta <BODY>, estos serán heredados por el resto de las etiquetas del documento. Esto es así siempre y cuando no se vuelvan a definir esos estilos en las siguientes etiquetas, en cuyo caso el estilo de la etiqueta más concreta será el que mande. Puede verse este detalle en la etiqueta <P>, que tiene definidos estilos que ya fueron definidos para <BODY>. Los estilos que se tienen en cuenta son los de la etiqueta <P>, que es más concreta.

Por último, ha de apreciarse los comentarios HTML que engloban toda la declaración de estilos: <!--Declaración de estilos-->. Estos comentarios se utilizan para que los navegadores antiguos, que no comprenden la sintaxis CSS, no incluyan ese texto en el cuerpo de la página. Si no se pusiera, los navegadores antiguos (por ejemplo Netscape 3) escribirían ese "feo código" en la página.

Estilo definido para todo un sitio web

Una de las características más potentes de la programación con hojas de estilos consiste en que, de una vez, podemos definir los estilos de todo un sitio web. Esto se consigue creando un archivo donde tan sólo se colocan las declaraciones de estilos de la página y enlazando todas las páginas del sitio con ese archivo. De este modo, todas las páginas comparten una misma declaración de estilos y, por tanto, si la se cambian, cambiarán todas las páginas. Con las ventajas añadidas de que se ahorra en líneas de código HTML (lo que reduce el peso del documento) y se evita la molestia de definir una y otra vez los estilos con el HTML, tal como se comentó anteriormente.

Veamos ahora cómo el proceso para incluir estilos con un archivo externo.

1. Creamos el archivo con la declaración de estilos

Es un archivo de texto normal, que puede tener cualquier extensión, aunque le podemos asignar la extensión .CSS para aclarar qué tipo de archivo es. El texto que se debe incluir debe ser escrito exclusivamente en sintaxis CSS, es decir, sería erróneo incluir código HTML en él: etiquetas y demás. Se puede ver un ejemplo a continuación.

```
P {  
  font-size : 12pt;  
  font-family : arial, helvetica;  
  font-weight : normal;  
}  
H1 {  
  font-size : 36pt;
```



```

font-family : verdana,arial;
text-decoration : underline;
text-align : center;
background-color : Teal;
}
TD {
font-size : 10pt;
font-family : verdana,arial;
text-align : center;
background-color : 666666;
}
BODY {
background-color : #006600;
font-family : arial;
color : White;
}

```

2. Enlazamos la página web con la hoja de estilos

Para ello, vamos a colocar la etiqueta <LINK> con los atributos

- **rel = "STYLESHEET"** indicando que el enlace es con una hoja de estilos
- **type = "text/css"** porque el archivo es de texto, en sintaxis CSS
- **href = "estilos.css"** indica el nombre del archivo fuente de los estilos

A continuación una página web entera que enlaza con la declaración de estilos anterior:

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
<link rel="STYLESHEET" type="text/css" href="estilos.css">
<title>P&acute;gina que lee estilos</title>
</head>
<body>
<h1>P&acute;gina que lee estilos</h1>
Esta p&acute;gina tiene en la cabecera la etiqueta necesaria para enlazar
con la hoja de estilos. Es muy
f&acute;cil.
<br>
<br>
<table width="300" cellspacing="2" cellpadding="2" border="0">
<tr>
<td>Esto est&acute; dentro de un TD, luego tiene estilo propio, declarado
en el fichero externo</td>
</tr>
<tr>
<td>La segunda fila del TD</td>

```

```
</tr>  
</table>  
</body>  
</html>
```

REGLAS DE IMPORTANCIA EN LOS ESTILOS

Los estilos se heredan de una etiqueta a otra, como se indicó anteriormente. Por ejemplo, si se tiene declarado en el <BODY> unos estilos, por lo general, estas declaraciones también afectarán a etiquetas que estén dentro de esta etiqueta, o lo que es lo mismo, dentro de todo el cuerpo.

En muchas ocasiones más de una declaración de estilos afecta a la misma porción de la página. Siempre se tiene en cuenta la declaración más particular. Pero las declaraciones de estilos se pueden realizar de múltiples modos y con varias etiquetas, también entre estos modos hay una jerarquía de importancia para resolver conflictos entre varias declaraciones de estilos distintas para una misma porción de página. Se puede ver a continuación esta jerarquía, primero ponemos las formas de declaración más generales, y por tanto menos respetadas en caso de conflicto:

- Declaración de estilos con archivo externo. (Para todo un sitio web)
- Declaración de estilos para toda la página. (Con la etiqueta <STYLE> en la cabecera de la página)
- Estilos definidos en una parte de la página. (Con la etiqueta <DIV>)
- Definidos en una etiqueta en concreto. (Utilizando el atributo style en la etiqueta en cuestión)
- Declaración de estilo para una porción pequeña del documento. (Con la etiqueta)

Ya se vio cómo incluir estilos en la página, de todas las maneras posibles y se hizo un repaso con la lista anterior. Ahora ya se está en condiciones de empezar a usar las hojas de estilo en cascada para mejorar tus páginas y aumentar la productividad de tu trabajo.

Otra manera de definir estilos en un archivo externo

Veamos ahora otra manera de importar una declaración externa de estilos CSS: @import url ("archivo_a_importar.css"), que se utiliza para importar unas declaraciones de estilos guardadas en la ruta que se indica entre paréntesis (las comillas son opcionales, pero los paréntesis son obligatorios, por lo menos, en Explorer).

Se debe incluir en la declaración de estilos global a una página, es decir entre las etiquetas <style type="text/css"> y </style>, que se colocan en la cabecera del documento.

Es importante señalar que la sentencia de importación del archivo CSS se debe escribir en la primera línea de la declaración de estilos, algo parecido al código siguiente.

```
<style type="text/css">
@import url ("estilo.css");
body{
background-color:#ffffcc;
}
</style>
```

El funcionamiento es el mismo que si se escribiera todo el archivo a importar dentro de las etiquetas de los estilos, con la salvedad de que, si se redefinen dentro del código HTML (entre las etiquetas </style>) estilos que habían quedado definidos en el archivo externo, los que se aplicarán serán los que hayamos redefinido.

Así, en el ejemplo anterior, aunque se hubiera definido en estilo CSS un color de fondo para la página, el color que prevalecería sería el definido a continuación de la importación: #ffffcc

La diferencia entre este tipo de importación del tipo y la que hemos visto anteriormente:

```
<link rel="stylesheet" type="text/css" href="hoja.css">
```

Es que @import url ("estilo.css") se suele utilizar cuando hay unas pautas básicas en el trabajo con los estilos (que se definen en un archivo a importar) y unos estilos específicos para cada página, que se definen a continuación, dentro del código HTML entre las etiquetas </style>, como es el caso del ejemplo visto anteriormente.

SINTAXIS CSS

Tal como se vio en los ejemplos la sintaxis es bastante sencilla y repetitiva. Como se ve a continuación:

- Para definir un estilo se utilizan atributos como font-size, text-decoration... seguidos de dos puntos y el valor que le deseemos asignar. Podemos definir un estilo a base de definir muchos atributos separados por punto y coma.

Ejemplo: font-size: 10pt; text-decoration: underline; color: black; (el último punto y coma de la lista de atributos es opcional)

- Para definir el estilo de una etiqueta se escribe la etiqueta seguida de la lista de atributos encerrados entre llaves.

Ejemplo:
H1{text-align: center; color:black}

- Los valores que se pueden asignar a los atributos de estilo se pueden ver en la tabla que se encuentra mas abajo. Muchos estos valores son unidades de medida, por ejemplo, el valor del tamaño de un margen o el tamaño de la fuente. Las unidades de medida son las siguientes:

Puntos	pt
Pulgadas	in
Centímetros	cm
pixels	px

Atributos de las hojas de estilo

Tanto para practicar en tu aprendizaje como para trabajar con las CSS lo mejor es disponer de una tabla donde se vean los distintos atributos y valores de estilos que podemos aplicarle a las páginas web.

Aquí se puede ver la tabla de los atributos CSS, se recomienda tenerla a mano cuando se utilizan las CSS.

Nombre del atributo	Posibles valores	Ejemplos
FUENTES – FONTS		
Color Sirve para indicar el color del texto. Lo admiten casi todas las etiquetas de HTML. No todos los nombres de colores son admitidos en el estándar, es aconsejable entonces utilizar el valor RGB.	Valor RGB o nombre de color	color: #009900; color: red;
font-size Sirve para indicar el tamaño de las fuentes de manera más rígida y con mayor exactitud.	xx-small x-small small medium large x-large xx-large Unidades de CSS	font-size:12pt; font-size: x-large;
font-family Con este atributo se indica la familia de tipografía del texto. Los primeros valores son genéricos, es decir, los exploradores las comprenden y utilizan las fuentes que el usuario tenga en su sistema. También se pueden definir con tipografías normales, como ocurría en html. Si el nombre de una fuente tiene espacios se utilizan comillas para que se entienda bien.	serif sans-serif cursive fantasy monospace Todas las fuentes habituales	font-family:arial,Helvetica; font-family: fantasy;
font-weight		

Nombre del atributo	Posibles valores	Ejemplos
Sirve para definir la anchura de los caracteres, o dicho de otra manera, para poner negrillas con total libertad. Normal y 400 son el mismo valor, así como bold y 700.	normal bold bolder lighter 100 200 300 400 500 600 700 800 900	font-weight:bold; font-weight: 200;
font-style Es el estilo de la fuente, que puede ser normal, itálica u oblicua. El estilo oblique es similar al italic.	normal italic oblique	font-style:normal; font-style: italic;
PÁRRAFOS – TEXT		
line-height El alto de una línea, y por tanto, el espaciado entre líneas. Es una de esas características que no se podían modificar utilizando HTML.	normal y unidades CSS	line-height: 12px; line-height: normal;
text-decoration Para establecer la decoración de un texto, es decir, si está subrayado, subrayado o tachado.	none [underline overline line-through]	text-decoration: none; text-decoration: underline;
text-align Sirve para indicar la alineación del texto. Es interesante destacar que las hojas de estilo permiten el justificado de texto, aunque recuerda que no tiene por que funcionar en todos los sistemas.	left right center justify	text-align: center; text-align: right;
text-indent Un atributo que sirve para hacer sangrado o márgenes en las páginas. Muy útil y novedosa.	Unidades CSS	text-indent: 10px; text-indent: 2in;
text-transform Nos permite transformar el texto, haciendo que tenga la primera letra en mayúsculas de todas las palabras, todo en mayúsculas o minúsculas.	capitalize uppercase lowercase none	text-transform: none; text-transform: capitalize;
FONDO - BACKGROUND		
Background-color Sirve para indicar el color de fondo de un elemento de la página.	Un color, con su nombre o su valor RGB	background-color: green; background-color: #000055;
Background-image Colocamos con este atributo una imagen de fondo en cualquier elemento de la página.	El nombre de la imagen con su camino relativo o absoluto	background-image: url(mármol.gif) ; background-image: url (http://www.x.com/fondo.gif)

Nombre del atributo	Posibles valores	Ejemplos
BOX – CAJA		
Margin-left Indicamos con este atributo el tamaño del margen a la izquierda	Unidades CSS	margin-left: 1cm; margin-left: 0,5in;
Margin-right Se utiliza para definir el tamaño del margen a la derecha	Unidades CSS	margin-right: 5%; margin-right: 1in;
Margin-top se indica con este atributo el tamaño del margen arriba de la página	Unidades CSS	margin-top: 0px; margin-top: 10px;
Margin-bottom Con el se indica el tamaño del margen en la parte de abajo de la página	Unidades CSS	margin-bottom: 0pt; margin-top: 1px
Padding-left Indica el espacio insertado, por la izquierda, entre el borde del elemento-continente y el contenido de este. Es parecido a el atributo cellpadding de las tablas. El espacio insertado tiene el mismo fondo que el fondo del elemento-continente.	Unidades CSS	padding-left: 0.5in; padding-left: 1px;
Padding-right Indica el espacio insertado, en este caso por la derecha, entre el borde del elemento-continente y el contenido de este. Es parecido a el atributo cellpadding de las tablas. El espacio insertado tiene el mismo fondo que el fondo del elemento-continente.	Unidades CSS	padding-right: 0.5cm; padding-right: 1pt;
Padding-top Indica el espacio insertado, por arriba, entre el borde del elemento-continente y el contenido de este.	Unidades CSS	padding-top: 10pt; padding-top: 5px;
Padding-bottom Indica el espacio insertado, en este caso por abajo, entre el borde del elemento-continente y el contenido de este.	Unidades CSS	padding-right: 0.5cm; padding-right: 1pt;
Border-color Para indicar el color del borde del elemento de la página al que se lo		

Nombre del atributo	Posibles valores	Ejemplos
aplicamos. Se puede poner colores por separado con los atributos border-top-color, border-right-color, border-bottom-color, border-left-color.	color RGB y nombre de color	border-color: red; border-color: #ffc0cb;
Border-style El estilo del borde, los valores significan: none=ningun borde, dotted=punteado (no parece funcionar), solid=solido, double=doble borde, y desde groove hasta outset son bordes con varios efectos 3D.	none dotted solid double groove ridge inset outset	border-style: solid; border-style: double;
border-width El tamaño del borde del elemento al que lo aplicamos.	Unidades CSS	border-width: 10px; border-width: 0.5in;
float Sirve para alinear un elemento a la izquierda o la derecha haciendo que el texto se agrupe alrededor de dicho elemento. Igual que el atributo align en imagenes en sus valores right y left.	none left right	float: right;
Clear Si este elemento tiene a su altura imágenes u otros elementos alineados a la derecha o la izquierda, con el atributo clear hacemos que se coloque en un lugar donde ya no tenga esos elementos al lado que indiquemos.	none right left	clear: right;

La especificación de estilos CSS es muy amplia, pero están la inmensa mayoría y por supuesto la selección de los más importantes.

TRUCOS AVANZADOS CON CSS

Definir estilos utilizando clases

Las clases sirven para crear definiciones de estilos que se pueden utilizar repetidas veces.

Una clase se puede definir entre las etiquetas <STYLE> (en la cabecera del documento), o en un archivo externo a la página. Para definir las se utiliza la siguiente sintaxis, un punto seguido del nombre de la clase y entre llaves los atributos de estilos deseados. De esta manera:

```
.nombredelaclase {atributo: valor; atributo2: valor2; ...}
```

Una vez se tiene una clase, se puede utilizar en cualquier etiqueta HTML. Para ello se utiliza el atributo class, poniéndole como valor el nombre de la clase, de esta forma:

```
<ETIQUETA class="nombredelaclase">
```

Ejemplo de la utilización de clases

```
<html>
<head>
<title>Ejemplo de la utilizaci&oacute;n de clases</title>
<STYLE type="text/css">
.fondonegroletrasblancas           {background-color:black;color:white;font-
size:12;font-family:arial}
.letrasverdes {color:#009900}
</STYLE>
</head>
<body>
<h1 class=letrasverdes>Titulo 1</h1>
<h1 class=fondonegroletrasblancas>Titulo 2</h1>
<p class=letrasverdes>
Esto es un p&aacute;rrafo con estilo de letras verdes</p>
<p class=fondonegroletrasblancas>
Esto es un p&aacute;rrafo con estilo de fondo negro y las letras blancas. Es
todo!</p>
</body>
</html>
```

Estilo en los enlaces

Una técnica muy habitual, que se puede realizar utilizando las hojas de estilo en cascada y no se podía en HTML, es la definición de estilos en los enlaces, quitándoles el subrayado o hacer enlaces en la misma página con distintos colores.

Para aplicar estilo a los enlaces se deben definir para los distintos tipos de enlaces que existen (visitados, activos...). Se Utiliza la siguiente sintaxis, en la declaración de estilos global de la página (<STYLE>) o del sitio (Definición en un archivo externo):

Enlaces normales

A:link {atributos}

Enlaces visitados

A:visited {atributos}

Enlaces activos (Los enlaces están activos en el preciso momento en que se pincha Sobre ellos)

A:active {atributos}

Enlaces hover (Cuando el ratón está encima de ellos, solo funciona en Iexplorer)

A:hover {atributos}

El atributo para definir enlaces sin subrayado es **text-decoration:none**, y para darles color es color:tu_color.

También se pueden definir el estilo de cada enlace en la propia etiqueta <A>, con el atributo style. De esta manera se puede hacer que determinados enlaces de la página se vean de manera distinta.

Ejemplo de estilos en enlaces

```
<html>
<head>
<title>Ejemplos de estilo en enlaces</title>
<STYLE type="text/css">
A:link {text-decoration:none;color:#0000cc;}
A:visited {text-decoration:none;color:#ffcc33;}
A:active {text-decoration:none;color:#ff0000;}
A:hover {text-decoration:underline;color:#999999;font-weight:bold}
</STYLE>
</head>
<body>
<a href="http://dominioinexistente.nofunciona.com">Enlace normal</a>
<br>
<br>
<a href="enlaces.html">Enlace visitado</a>
Pulsar este enlace para verlo activo,
poner el ratón por encima para que cambie.
</body>
</html>
```

URL como valor de un atributo:

Determinados atributos de estilos, como **background-image** necesitan una URL como valor, para indicarlas podemos definir tanto caminos relativos como absolutos. Así pues, podemos indicar la URL de la imagen de fondo de estas dos maneras:

background-image: url(fondo.gif) En caso de que la imagen esté en el mismo directorio que la página.

background-image: url(http://www.pagina.com/directorio/fondo.gif)

Ocultar estilos en navegadores antiguos

En caso de definir dentro de la etiqueta <STYLE> unas declaraciones de estilos se debe asegurar que estas no se imprimirán en la página web con navegadores antiguos. Para evitarlo debemos ocultar con comentarios HTML (<!-- esto es un comentario -->) todo lo que hay dentro de la etiqueta <STYLE>.

INTRODUCCIÓN A JAVASCRIPT

Javascript es un lenguaje de programación utilizado para crear pequeños programitas encargados de realizar acciones dentro del ámbito de una página web. Con Javascript se pueden crear efectos especiales en las páginas y definir interactividades con el usuario. El navegador del cliente es el encargado de interpretar las instrucciones Javascript y ejecutarlas para realizar estos efectos e interactividades, de modo que el mayor recurso, y tal vez el único, con que cuenta este lenguaje es el propio navegador.

Javascript es el siguiente paso, después del HTML, que puede dar un programador de la web que decida mejorar sus páginas y la potencia de sus proyectos. Es un lenguaje de programación bastante sencillo y pensado para hacer las cosas con rapidez, a veces con ligereza. Incluso las personas que no tengan una experiencia previa en la programación podrán aprender este lenguaje con facilidad y utilizarlo en toda su potencia con sólo un poco de práctica.

Entre las acciones típicas que se pueden realizar en Javascript tenemos dos vertientes. Por un lado los efectos especiales sobre páginas web, para crear contenidos dinámicos y elementos de la página que tengan movimiento, cambien de color o cualquier otro dinamismo. Por el otro, javascript nos permite ejecutar instrucciones como respuesta a las acciones del usuario, con lo que podemos crear páginas interactivas con programas como calculadoras, agendas, o tablas de cálculo.

Javascript es un lenguaje con muchas posibilidades, permite la programación de pequeños scripts, pero también de programas más grandes, orientados a objetos, con funciones, estructuras de datos complejas, etc. Toda esta potencia de Javascript se pone

a disposición del programador, que se convierte en el verdadero dueño y controlador de cada cosa que ocurre en la página.

ALGO DE HISTORIA SOBRE JAVASCRIPT

En Internet se han creado multitud de servicios para realizar muchos tipos de comunicaciones, como correo, charlas, búsquedas de información, etc. Pero ninguno de estos servicios se ha desarrollado tanto como el Web..

El web es un sistema Hipertexto, una cantidad desmesurada de textos que contienen enlaces que relacionan cada una de las unidades básicas donde podemos encontrar información, las páginas web. En un principio, para diseñar este sistema de páginas con enlaces se pensó en un lenguaje que permitiese presentar cada una de estas informaciones junto con unos pequeños estilos, este lenguaje fue el HTML, que luego se vería que no cumplió todos los objetivos para los que fue diseñado, pero eso es otro tema.

El caso es que HTML no es suficiente para realizar todas las acciones que se pueden llegar a necesitar en una página web. Esto es debido a que conforme fue creciendo el web y sus distintos usos se fueron complicando las páginas y las acciones que se querían realizar a través de ellas. El HTML se había quedado corto para definir todas estas nuevas funcionalidades, ya que sólo sirve para presentar el texto en una página, definir su estilo y poco más.

El primer ayudante para cubrir las necesidades que estaban surgiendo fue Java, a través de la tecnología de los Applets, que son pequeños programas que se incrustan en las páginas web y que pueden realizar las acciones asociadas a los programas de propósito general. La programación de Applets fue un gran avance y Netscape, por aquel entonces el navegador más popular, había roto la primera barrera del HTML al hacer posible la programación dentro de las páginas Web. No cabe duda que la aparición de los Applets supuso un gran avance en la historia del Web, pero no ha sido una tecnología definitiva y muchas otras han seguido implementando el camino que comenzó con ellos.

Netscape, después de hacer sus navegadores compatibles con los Applets, comenzó a desarrollar un lenguaje de programación al que llamó LiveScript que permitiese crear pequeños programas en las páginas y que fuese mucho más sencillo de utilizar que Java. De modo que el primer Javascript se llamó LiveScript, pero no duró mucho ese nombre, pues antes de lanzar la primera versión del producto se forjó una alianza con Sun Microsystems, creador de Java, para desarrollar en conjunto ese nuevo lenguaje.

La alianza hizo que Javascript se diseñara como un hermano pequeño de Java, solamente útil dentro de las páginas Web y mucho más fácil de utilizar, de modo que cualquier persona, sin conocimientos de programación pudiese adentrarse en el lenguaje y utilizarlo a sus anchas. Además, para programar Javascript no es necesario un kit de desarrollo, ni compilar los scripts, ni realizarlos en Archivos externos al código HTML, como ocurría con los applets.

Netscape 2.0 fue el primer navegador que entendía Javascript y su estela fue seguida por los navegadores de la compañía Microsoft a partir de la versión 3.0.

DIFERENCIAS ENTRE JAVA Y JAVASCRIPT

Debe quedar claro que Javascript no tiene nada que ver con Java, salvo en sus orígenes, como se ha podido leer hace unas líneas. Actualmente son productos totalmente distintos y no guardan entre sí más relación que la sintaxis idéntica y poco más. Algunas diferencias entre estos dos lenguajes son las siguientes:

- **Compilador.** Para programar en Java necesitamos un Kit de desarrollo y un compilador. Sin embargo, Javascript no es un lenguaje que necesite que sus programas se compilen, sino que éstos se interpretan por parte del navegador cuando éste lee la página.
- **Orientado a objetos.** Java es un lenguaje de programación orientado a objetos. (Más tarde veremos que quiere decir orientado a objetos, para el que no lo sepa todavía) Javascript no es orientado a objetos, esto quiere decir que podremos programar sin necesidad de crear clases, tal como se realiza en los lenguajes de programación estructurada como C o Pascal.
- **Propósito.** Java es mucho más potente que Javascript, esto es debido a que Java es un lenguaje de propósito general, con el que se pueden hacer aplicaciones de lo más variado, sin embargo, con Javascript sólo podemos escribir programas para que se ejecuten en páginas Web.
- **Estructuras fuertes.** Java es un lenguaje de programación fuertemente tipado, esto quiere decir que al declarar una variable tendremos que indicar su tipo y no podrá cambiar de un tipo a otro automáticamente. Por su parte Javascript no tiene esta característica, y podemos meter en una variable la información que deseemos, independientemente del tipo de ésta. Además, podremos cambiar el tipo de información de una variable cuando queramos.
- **Otras características.** Como vemos Java es mucho más complejo, aunque también más potente, robusto y seguro. Tiene más funcionalidades que Javascript y las diferencias que los separan son lo suficientemente importantes como para distinguirlos fácilmente.

USOS DE JAVASCRIPT

A continuación se ven brevemente algunos usos de este lenguaje que podemos encontrar en el Web para hacernos una idea de las posibilidades que tiene.

Para empezar, podemos ver páginas como la página de SEAT, llena de efectos super interesantes sobre Javascript, que llegan a asemejarse a la tecnología Flash. En esta misma vertiente de uso de Javascript podemos encontrar muchas páginas, por ejemplo la página Guiarte Multimedia, que realiza una animación Javascript para presentar a la empresa dueña de la página Web.

Por otro lado, podemos encontrar dentro de Internet muchas aplicaciones de Javascript mucho más serias, que hacen que una página Web se convierta en un verdadero programa interactivo de gestión de cualquier recurso. Por ejemplo podemos ver una página que consiste en un test de preguntas, que recoge los resultados y los envía a su evaluador. La dirección es <http://www.geocities.com/Athens/Oracle/3391/>. También se pueden ver más ejemplos de estos dentro de cualquier página un poco compleja, si nos pasamos por un sitio que tenga una calculadora o un convertidor de divisas, veremos que en muchos casos se han realizado con Javascript.

En realidad es mucho más habitual encontrar Javascript para realizar efectos simples sobre páginas Web, o no tan simples, como pueden ser rollovers (que cambie una imagen al pasar el ratón por encima), navegadores desplegados, apertura de ventanas secundarias, etc. Nos atrevemos a decir que este lenguaje es realmente útil para estos casos, pues estos típicos efectos tienen la complejidad justa para ser implementados en cuestión de minutos sin posibilidad de errores. Las páginas de Desarrollo Web son un ejemplo de páginas que utilizan Javascript para realizar multitud de acciones sin que estas sean demasiado complicadas, que carguen la página o que den lugar a errores en distintas plataformas.

Para programar en Javascript se necesita básicamente lo mismo que para programar páginas Web con HTML. Un editor de textos y un navegador compatible con Javascript. Un usuario de Windows posee de salida todo lo necesario para poder programar en Javascript, puesto que dispone dentro de su instalación típica de sistema operativo, de un editor de textos, el Bloc de notas, y de un navegador: Internet Explorer.

Usuarios de otros sistemas pueden encontrar en Internet fácilmente las herramientas necesarias para comenzar en páginas de descarga de software como Tucows.

Aunque el Bloc de Notas es suficiente para empezar, tal vez sea muy útil contar con otros programas que nos ofrecen mejores prestaciones a la hora de escribir las líneas de código. Estos editores avanzados tienen algunas ventajas como que colorean los códigos de los scripts, permiten trabajar con varios documentos simultáneamente, tienen ayudas, etc. Entre otros queremos destacar el Home Site o UltraEdit.

VERSIONES DE NAVEGADORES Y DE JAVASCRIPT

También resulta apropiado introducir las distintas versiones de Javascript que existen y que han evolucionado en conjunto con las versiones de navegadores. El lenguaje ha ido avanzando durante sus años de vida e incrementando sus capacidades. En un principio podía realizar muchas cosas en la página Web, pero tenía pocas instrucciones para crear efectos especiales. Con el tiempo también el HTML ha avanzado y se han creado nuevas características como las capas, que permiten tratar y planear los documentos de manera distinta. Javascript ha avanzado también y para manejar todas estas nuevas características se han creado nuevas instrucciones y recursos.

- Javascript 1: nació con el Netscape 2.0 y soportaba gran cantidad de instrucciones y funciones, casi todas las que existen ahora ya se introdujeron en el primer estándar.
- Javascript 1.1: Es la versión de Javascript que se diseñó con la llegada de los navegadores 3.0. Implementaba poco más que su anterior versión, como por ejemplo el tratamiento de imágenes dinámicamente y la creación de arrays.
- Javascript 1.2: La versión de los navegadores 4.0. Esta tiene como desventaja que es un poco distinta en plataformas Microsoft y Netscape, ya que ambos navegadores crecieron de distinto modo y estaban en plena lucha por el mercado.
- Javascript 1.3: Versión que implementan los navegadores 5.0. En esta versión se han limado algunas diferencias y asperezas entre los dos navegadores.
- Javascript 1.5: Versión actual, en el momento de escribir estas líneas, que implementa Netscape 6.
- Por su parte, Microsoft también ha evolucionado hasta presentar su versión 5.5 de JScript (así llaman al javascript utilizado por los navegadores de Microsoft).

EFFECTOS RÁPIDOS.

Fecha de última modificación

Primero vamos un sencillo script para mostrar la fecha de la última actualización del documento. A veces es muy interesante saber en qué fecha se realizó la última modificación y da una imagen de página actualizada, siempre y cuando se actualice de verdad. Además, entre actualización y actualización no debemos cambiar la fecha, pues se cambia ella sola al grabar el documento.

```
<script> document.write(document.lastModified) </script>
```

Estas líneas se deben introducir dentro del cuerpo de la página en el lugar donde queramos que aparezca la fecha de última actualización. Un detalle a destacar es que la fecha aparece en un formato un poco raro, indicando la hora y otros atributos de la misma. Vemos que es sencillo y rápido de hacer. Es el ejemplo típico de la utilidad de Javascript y su facilidad de uso.

Botón de volver

Otro ejemplo rápido se puede ver a continuación. Se trata de un botón para volver hacia atrás, como el que tenemos en la barra de herramientas del navegador.

```
<input type=button value=Atrás onclick="history.go(-1)">
```

Como diferencia con el ejemplo anterior hay que destacar que en este caso la instrucción Javascript se encuentra dentro de un atributo de HTML, onclick, que indica que esa instrucción se tiene que ejecutar como respuesta a la pulsación del botón.

EL LENGUAJE JAVASCRIPT

Lo más importante y básico que podemos destacar en este momento es que la programación de Javascript se realiza dentro del propio documento HTML. Esto quiere decir que en la página se mezclan los dos lenguajes de programación, y para que estos dos lenguajes se puedan mezclar sin problemas se han de incluir unos delimitadores que separan las etiquetas HTML de las instrucciones Javascript. Estos delimitadores son las etiquetas `<SCRIPT>` y `</SCRIPT>`. Todo el código Javascript que pongamos en la página ha de ser introducido entre estas dos etiquetas.

En una misma página podemos introducir varios scripts, cada uno que podría introducirse dentro de unas etiquetas `<SCRIPT>` distintas. La colocación de estos scripts es indiferente, en un principio nos da igual donde colocarlos, pero en determinados casos esta colocación si que será muy importante. En cada caso, y llegado el momento se informará de ello convenientemente.

También se puede escribir Javascript dentro de determinados atributos de la página, como el atributo onclick. Estos atributos están relacionados con las acciones del usuario y se llaman manejadores de eventos.

Existen dos maneras de ejecutar scripts en la página. La primera de estas maneras se trata de ejecución directa de scripts, la segunda es una ejecución como respuesta a la acción de un usuario. Veremos ahora cada una de ellas.

Ejecución directa

Es el método de ejecutar scripts más básico. En este caso se incluyen las instrucciones dentro de la etiqueta <SCRIPT>, tal como hemos comentado anteriormente. Cuando el navegador lee la página y encuentra un script va interpretando las líneas de código y las va ejecutando una después de otra. Llamamos a esta manera ejecución directa pues cuando se lee la página se ejecutan directamente los scripts.

Respuesta a un evento

Es la otra manera de ejecutar scripts, pero antes de verla debemos hablar sobre los eventos. Los eventos son acciones que realiza el usuario. Los programas como Javascript están preparados para atrapar determinadas acciones realizadas, en este caso sobre la página, y realizar acciones como respuesta. De este modo se pueden realizar programas interactivos, ya que controlamos los movimientos del usuario y respondemos a ellos. Existen muchos tipos de eventos distintos, por ejemplo la pulsación de un botón, el movimiento del ratón o la selección de texto de la página.

Las acciones que queremos realizar como respuesta a un evento se han de indicar dentro del mismo código HTML, pero en este caso se indican en atributos HTML que se colocan dentro de la etiqueta que queremos que responda a las acciones del usuario. En el capítulo donde vimos algún ejemplo rápido ya comprobamos que si queríamos que un botón realizase acciones cuando se pulsase sobre el, debíamos indicarlo dentro del atributo onclick del botón.

Comprobamos pues que se puede introducir código Javascript dentro de determinados atributos de las etiquetas HTML. Veremos más adelante este tipo de ejecución en profundidad y los tipos de eventos que existen.

Ocultar scripts en navegadores antiguos

Ya hemos visto que Javascript se implementó a partir de Netscape 2.0 e Internet Explorer 3.0, incluso hay navegadores que funcionan en sistemas donde sólo se puede visualizar texto y por lo tanto determinadas tecnologías, como este lenguaje, están fuera de su alcance. Así pues, no todos los navegadores del Web comprenden Javascript. En los casos en los que no se interpretan los scripts, los navegadores asumen que el código de éstos es texto de la propia página Web y como consecuencia, presentan los scripts en la página Web como si de texto normal se tratara. Para evitar que el texto de los scripts se escriba en la página cuando los navegadores no los entienden se tienen que ocultar las instrucciones con comentarios HTML (<!--comentario HTML -->). Veamos con un ejemplo cómo se han de ocultar los scripts.

```
<SCRIPT>  
<!--  
Código Javascript  
/-->  
</SCRIPT>
```


Vemos que el inicio del comentario HTML es idéntico a cómo lo conocemos en el HTML, pero el cierre del comentario presenta una particularidad, que empieza por doble barra inclinada. Esto es debido a que el final del comentario contiene varios caracteres que Javascript reconoce como operadores y al tratar de analizarlos lanza un mensaje de error de sintaxis. Para que Javascript no lance un mensaje de error se coloca antes del comentario HTML esa doble barra, que no es más que un comentario Javascript, que se conocerá más adelante cuando hablemos de sintaxis.

El inicio del comentario HTML no es necesario comentarlo con la doble barra, dado que Javascript entiende bien que simplemente se pretende ocultar el código. Una aclaración a este punto: si se pusieran las dos barras en esta línea, se verían en navegadores antiguos por estar fuera de los comentarios HTML. Las etiquetas `<SCRIPT>` no las entienden los navegadores antiguos, por lo tanto no las interpretan, tal como hacen con cualquier etiqueta que desconocen.

```
<NOSCRIPT>
```

Existe la posibilidad de indicar un texto alternativo para los navegadores que no entienden Javascript, para informarles de que en ese lugar debería ejecutarse un script y que la página no está funcionando al 100% de sus capacidades. También podemos sugerir a los visitantes que actualicen su navegador a una versión compatible con el lenguaje. Para ello utilizamos la etiqueta `<NOSCRIPT>` y entre esta etiqueta y su correspondiente de cierre podemos colocar el texto alternativo al script.

```
<SCRIPT>
código javascript
</SCRIPT>
<NOSCRIPT>
Este navegador no comprende los scripts que se están ejecutando, debes
actualizar tu versión de navegador a una más reciente.
<br><br>
<a href=http://netscape.com>Netscape</a>.<br>
<a href=http://microsoft.com>Microsoft</a>.
</NOSCRIPT>
```

Un par de notas adicionales sobre cómo colocar scripts en páginas web.

- Lenguaje que estamos utilizando

La etiqueta `<SCRIPT>` tiene un atributo que sirve para indicar el lenguaje que estamos utilizando, así como la versión de este. Por ejemplo, podemos indicar que estamos programando en Javascript 1.2 o Visual Basic Script, que es otro lenguaje para programar scripts en el navegador cliente que sólo es compatible con Internet Explorer.

El atributo en cuestión es `language` y lo más habitual es indicar simplemente el lenguaje con el que se han programado los scripts. El lenguaje por defecto es Javascript, por lo que si no utilizamos este atributo, el navegador entenderá que el

lenguaje con el que se está programando es Javascript. Un detalle donde se suele equivocar la gente sin darse cuenta es que language se escribe con dos -g- y no con -g- y con -j- como en castellano.

```
<SCRIPT LANGUAGE=javascript>
```

- Archivos externos de Javascript

Otra manera de incluir scripts en páginas web, implementada a partir de Javascript 1.1, es incluir archivos externos donde se pueden colocar muchas funciones que se utilicen en la página. Los Archivos suelen tener extensión .js y se incluyen de esta manera.

```
<SCRIPT language=javascript src="archivo_externo.js">  
//estoy incluyendo el fichero "archivo_externo.js"  
</SCRIPT>
```

Dentro de las etiquetas <SCRIPT> se puede escribir cualquier texto y será ignorado por el navegador, sin embargo, los navegadores que no entienden el atributo SRC tendrán a este texto por instrucciones, por lo que es aconsejable poner un comentario Javascript antes de cada línea con el objetivo de que no respondan con un error.

El archivo que incluimos (en este caso archivo_externo.js) debe contener tan solo sentencias Javascript. No debemos incluir código HTML de ningún tipo, ni tan siquiera las etiquetas </SCRIPT> y </SCRIPT>.

SINTAXIS JAVASCRIPT

El lenguaje Javascript tiene una sintaxis muy parecida a la de Java por estar basado en él. También es muy parecida a la del lenguaje C, de modo que si el lector conoce alguno de estos dos lenguajes se podrá manejar con facilidad con el código.

- **Comentarios**

Un comentario es una parte de código que no es interpretada por el navegador y cuya utilidad radica en facilitar la lectura al programador. El programador, a medida que desarrolla el script, va dejando frases o palabras sueltas, llamadas comentarios, que le ayudan a él o a cualquier otro a leer más fácilmente el script a la hora de modificarlo o depurarlo.

Existen dos tipos de comentarios en el lenguaje. Uno de ellos, la doble barra, sirve para comentar una línea de código. El otro comentario lo podemos utilizar para comentar varias líneas y se indica con los signos /* para empezar el comentario y */ para terminarlo.

Ejemplo:

```
<SCRIPT>  
//Este es un comentario de una línea  
/*Este comentario se puede extender  
por varias líneas.  
Las que quieras*/  
</SCRIPT>
```

- **Mayúsculas y minúsculas**

En javascript se han de respetar las mayúsculas y las minúsculas. Si nos equivocamos al utilizarlas el navegador responderá con un mensaje de error de sintaxis. Por convención los nombres de las cosas se escriben en minúsculas, salvo que se utilice un nombre con más de una palabra, pues en ese caso se escribirán con mayúsculas las iniciales de las palabras siguientes a la primera. También se puede utilizar mayúscula en las iniciales de las primeras palabras en algunos casos, como los nombres de las clases, aunque ya veremos más adelante cuáles son estos casos y qué son las clases.

- **Separación de instrucciones**

Las distintas instrucciones que contienen nuestros scripts se han de separar convenientemente para que el navegador no indique los correspondientes errores de sintaxis. Javascript tiene dos maneras de separar instrucciones. La primera es a través del carácter punto y coma (;) y la segunda es a través de un salto de línea.

Por esta razón las sentencias Javascript no necesitan acabar en punto y coma a no ser que coloquemos dos instrucciones en la misma línea.

No es una mala idea, de todos modos, acostumbrarse a utilizar el punto y coma después de cada instrucción pues otros lenguajes como Java o C obligan a utilizarlas y nos estaremos acostumbrando a realizar una sintaxis más parecida a la habitual en entornos de programación avanzados.

- **Variables Javascript**

Una variable es un espacio en memoria donde se almacena un dato, un espacio donde podemos guardar cualquier tipo de información que necesitemos para realizar las acciones de nuestros programas. Por ejemplo, si nuestro programa realiza sumas, será muy normal que guardemos en variables los distintos sumandos que participan en la operación y el resultado de la suma. El efecto sería algo parecido a esto.

```
sumando1 = 23  
sumando2 = 33  
suma = sumando1 + sumando2
```

En este ejemplo tenemos tres variables, `sumando1`, `sumando2` y `suma`, donde guardamos el resultado. Vemos que su uso para nosotros es como si tuviésemos un apartado donde guardar un dato y que se pueden acceder a ellos con sólo poner su nombre.

Los nombres de las variables han de construirse con caracteres alfanuméricos y el carácter subrayado (`_`). Aparte de esta, hay una serie de reglas adicionales para construir nombres para variables. La más importante es que tienen que comenzar por un carácter alfabético o el subrayado. No podemos utilizar caracteres raros como el signo `+`, un espacio o un `$`. Nombres admitidos para las variables podrían ser

```
Edad  
paisDeNacimiento  
_nombre
```

También hay que evitar utilizar nombres reservados como variables, por ejemplo no podremos llamar a nuestra variable palabras como `return` o `for`, que ya veremos que son utilizadas para estructuras del propio lenguaje. Veamos ahora algunos nombres de variables que no están permitidos utilizar

```
12meses  
tu nombre  
return  
pe%pe
```

- **Declaración de variables**

Declarar variables consiste en definir y de paso informar al sistema de que se va a utilizar una variable. Es una costumbre habitual en los lenguajes de programación el definir las variables que se van a usar en los programas y para ello, se siguen unas reglas estrictas. Pero javascript se salta muchas reglas por ser un lenguaje un tanto libre a la hora de programar y uno de los casos en los que otorga un poco de libertad es a la hora de declarar las variables, ya que no estamos obligados a hacerlo, al contrario de lo que pasa en la mayoría de los lenguajes de programación.

De todos modos, es aconsejable declarar las variables, además de una buena costumbre y para ello Javascript cuenta con la palabra `var`. Como es lógico, se utiliza esa palabra para definir la variable antes de utilizarla.

```
var operando1  
var operando2
```

También se puede asignar un valor a la variable cuando se está declarando

```
var operando1 = 23  
var operando2 = 33
```

También se permite declarar varias variables en la misma línea, siempre que se separen por comas.

```
var operando1,operando2
```

- **Ámbito de las variables**

Se le llama ámbito de las variables al lugar donde estas están disponibles. Por lo general, cuando declaramos una variable hacemos que esté disponible en el lugar donde se ha declarado, esto ocurre en todos los lenguajes de programación y como javascript se define dentro de una página web, las variables que declaremos en la página estarán accesibles dentro de ella. De este modo, no podremos acceder a variables que hayan sido definidas en otra página. Este es el ámbito más habitual de una variable y le llamaremos a este tipo de variables globales a la página, aunque no será el único, ya que también podremos declarar variables en lugares más acotados.

- **Variables globales**

Como se ha dicho, las variables globales son las que están declaradas en el ámbito más amplio posible, que en Javascript es una página web. Para declarar una variable global a la página simplemente se hace en un script, con la palabra var.

```
<SCRIPT>  
var variableGlobal  
</SCRIPT>
```

Las variables globales son accesibles desde cualquier lugar de la página, es decir, desde el script donde se han declarado y todos los demás scripts de la página, incluidos los manejadores de eventos, como el onclick, que ya vimos que se podía incluir dentro de determinadas etiquetas HTML.

- **Variables locales**

También se pueden declarar variables en lugares más acotados, como por ejemplo una función. A estas variables les llamaremos locales. Cuando se declaren variables locales sólo podremos acceder a ellas dentro del lugar donde se ha declarado, es decir, si la habíamos declarado en una función solo podremos acceder a ella cuando estemos en esa función.

Las variables pueden ser locales a una función, pero también pueden ser locales a otros ámbitos, como por ejemplo un bucle. En general, son ámbitos locales cualquier lugar acotado por llaves.

```
<SCRIPT>  
function miFuncion (){  
    var variableLocal  
}  
</SCRIPT>
```

En el script anterior se ha declarado una variable dentro de una función, por lo que esa variable sólo tendrá validez dentro de la función. Se pueden ver cómo se utilizan las llaves para acotar el lugar donde está definida esa función o su ámbito.

No hay problema en declarar una variable local con el mismo nombre que una global, en este caso la variable global será visible desde toda la página, excepto en el ámbito donde está declarada la variable local ya que en este sitio ese nombre de variable está ocupado por la local y es ella quien tiene validez.

```
<SCRIPT>
var numero = 2
function miFuncion (){
  var numero = 19
  document.write(numero) //imprime 19
}
document.write(numero) //imprime 2
</SCRIPT>
```

Un consejo para los principiantes podría ser no declarar variables con los mismos nombres, para que nunca haya lugar a confusión sobre qué variable es la que tiene validez en cada momento.

- **Diferencias entre utilizar var o no**

Como se ha dicho, en Javascript tenemos libertad para declarar o no las variables con la palabra var, pero los efectos que conseguiremos en cada caso serán distintos. En concreto, cuando utilizamos var estamos haciendo que la variable que estamos declarando sea local al ámbito donde se declara. Por otro lado, si no utilizamos la palabra var para declarar una variable, ésta será global a toda la página, sea cual sea el ámbito en el que haya sido declarada.

En el caso de una variable declarada en la página web, fuera de una función o cualquier otro ámbito más reducido, nos es indiferente si se declara o no con var, desde un punto de vista funcional. Esto es debido a que cualquier variable declarada fuera de un ámbito es global a toda la página. La diferencia se puede apreciar en una función por ejemplo, ya que si utilizamos var la variable será local a la función y si no lo utilizamos, la variable será global a la página. Esta diferencia es fundamental a la hora de controlar correctamente el uso de las variables en la página, ya que si no lo hacemos en una función podríamos sobrescribir el valor de una variable, perdiendo el dato que pudiera contener previamente.

```
<SCRIPT>
var numero = 2
function miFuncion (){
  numero = 19
  document.write(numero) //imprime 19
}
```

```
document.write(numero) //imprime 2
//llamamos a la función
miFuncion()
document.write(numero) //imprime 19
</SCRIPT>
```

En este ejemplo, tenemos una variable global a la página llamada número, que contiene un 2. También tenemos una función que utiliza la variable numero sin haberla declarado con var, por lo que la variable numero de la función será la misma variable global numero declarada fuera de la función. En una situación como esta, al ejecutar la función se sobrescribirá la variable numero y el dato que había antes de ejecutar la función se perderá.

- **¿Qué podemos guardar en variables?**

En una variable podemos introducir varios tipos de información, por ejemplo texto, números enteros o reales, etc. A estas distintas clases de información se les conoce como tipos de datos. Cada uno tiene características y usos distintos, veamos cuáles son los tipos de datos de Javascript.

Números

Para empezar tenemos el tipo numérico, para guardar números como 9 o 23.6

Cadenas

El tipo cadena de carácter guarda un texto. Siempre que escribamos una cadena de caracteres debemos utilizar las comillas ("").

Boleanos

También contamos con el tipo boleano, que guarda una información que puede valer si (true) o no (false).

Por último sería relevante señalar aquí que nuestras variables pueden contener cosas más complicadas, como podría ser un objeto, una función, o vacío (null) pero ya lo veremos más adelante.

En realidad nuestras variables no están forzadas a guardar un tipo de datos en concreto y por lo tanto no especificamos ningún tipo de datos para una variable cuando la estamos declarando. Podemos introducir cualquier información en una variable de cualquier tipo, incluso podemos ir cambiando el contenido de una variable de un tipo a otro sin ningún problema. Vamos a ver esto con un ejemplo.

```
var nombre_ciudad = "Valencia"
var revisado = true
nombre_ciudad = 32
revisado = "no"
```

Esta ligereza a la hora de asignar tipos a las variables puede ser una ventaja en un principio, sobretodo para personas inexpertas, pero a la larga puede ser fuente de errores

ya que dependiendo del tipo que son las variables se comportarán de un modo u otro y si no controlamos con exactitud el tipo de las variables podemos encontrarnos sumando un texto a un número. Javascript operará perfectamente, y devolverá un dato, pero en algunos casos puede que no sea lo que estábamos esperando. Así pues, aunque tenemos libertad con los tipos, esta misma libertad nos hace estar más atentos a posibles desajustes difíciles de detectar a lo largo de los programas. Veamos lo que ocurriría en caso de sumar letras y números.

```
var sumando1 = 23
var sumando2 = "33"
var suma = sumando1 + sumando2
document.write(suma)
```

Este script nos mostraría en la página el texto 2333, que no se corresponde con la suma de los dos números, sino con su concatenación, uno detrás del otro.

- **Tipos de datos en Javascript**

En nuestros scripts vamos a manejar variables diversas clases de información, como textos o números. Cada una de estas clases de información son los tipos de datos. Javascript distingue entre tres tipos de datos y todas las informaciones que se puedan guardar en variables van a estar encajadas en uno de estos tipos de datos. Veamos detenidamente cuáles son estos tres tipos de datos.

Tipo de datos numérico

En este lenguaje sólo existe un tipo de datos numérico, al contrario que ocurre en la mayoría de los lenguajes más conocidos. Todos los números son por tanto del tipo numérico, independientemente de la precisión que tengan o si son números reales o enteros. Los números enteros son números que no tienen coma, como 3 o 339. Los números reales son números fraccionarios, como 2.69 o 0.25, que también se pueden escribir en notación científica, por ejemplo 2.482e12.

Con Javascript también podemos escribir números en otras bases, como la hexadecimal. Las bases son sistemas de numeración que utilizan más o menos dígitos para escribir los números. Existen tres bases con las que podemos trabajar

- Base 10, es el sistema que utilizamos habitualmente, el sistema decimal. Cualquier número, por defecto, se entiende que está escrito en base 10.
- Base 8, también llamado sistema octal, que utiliza dígitos del 0 al 7. Para escribir un número en octal basta con escribir ese número precedido de un 0, por ejemplo 045.
- Base 16 o sistema hexadecimal, es el sistema de numeración que utiliza 16 dígitos, los comprendidos entre el 0 y el 9 y las letras de la A a la F, para los dígitos que faltan. Para escribir un número en hexadecimal debemos escribirlo precedido de un cero y una equis, por ejemplo 0x3EF.

Tipo boleano

El tipo bolean, boolean en inglés, sirve para guardar un si o un no o dicho de otro modo, un verdadero o un falso. Se utiliza para realizar operaciones lógicas, generalmente para realizar acciones si el contenido de una variable es verdadero o falso.

Si una variable es verdadero entonces Ejecuto unas instrucciones Si no Ejecuto otras

Los dos valores que pueden tener las variables boleanas son true o false.

```
miBoleana = true  
miBoleana = false
```

Tipo de datos cadena de caracteres

El último tipo de datos es el que sirve para guardar un texto. Javascript sólo tiene un tipo de datos para guardar texto y en el se pueden introducir cualquier número de caracteres. Un texto puede estar compuesto de números, letras y cualquier otro tipo de caracteres y signos. Los textos se escriben entre comillas, dobles o simples.

```
miTexto = "Pepe se va a pescar"  
miTexto = '23%%$ Letras & *--*'
```

Todo lo que se coloca entre comillas, como en los ejemplos anteriores es tratado como una cadena de caracteres independientemente de lo que coloquemos en el interior de las comillas. Por ejemplo, en una variable de texto podemos guardar números y en ese caso tenemos que tener en cuenta que las variables de tipo texto y las numéricas no son la misma cosa y mientras que las de numéricas nos sirven para hacer cálculos matemáticos las de texto no.

Caracteres de escape en cadenas de texto.

Hay una serie de caracteres especiales que sirven para expresar en una cadena de texto determinados controles como puede ser un salto de línea o un tabulador. Estos son los caracteres de escape y se escriben con una notación especial que comienza por una contra barra (una barra inclinada al revés de la normal '\') y luego se coloca el código del carácter a mostrar.

Un carácter muy común es el salto de línea, que se consigue escribiendo \n. Otro carácter muy habitual es colocar unas comillas, pues si colocamos unas comillas sin su carácter especial nos cerrarían las comillas que colocamos para iniciar la cadena de caracteres. Las comillas las tenemos que introducir entonces con \" o \' (comillas dobles o simples). Existen otros caracteres de escape, que veremos en la tabla de abajo más resumidos, aunque también hay que destacar como carácter habitual el que se utiliza para escribir una contrabarra, para no confundirla con el inicio de un carácter de escape, que es la doble contrabarra \\.

Tabla con todos los caracteres de escape

<i>Salto de línea:</i>	<code>\n</code>
<i>Comilla simple:</i>	<code>'</code>
<i>Comilla doble:</i>	<code>"</code>
<i>Tabulador:</i>	<code>\t</code>
<i>Retorno de carro:</i>	<code>\r</code>
<i>Avance de página:</i>	<code>\f</code>
<i>Retroceder espacio:</i>	<code>\b</code>
<i>Contrabarra:</i>	<code>\\</code>

Operadores Javascript

Al desarrollar programas en cualquier lenguaje se utilizan los operadores. Éstos sirven para hacer los cálculos y operaciones necesarios para llevar a cabo sus objetivos. Un programa que no realiza operaciones solo se puede limitar a hacer siempre lo mismo, es el resultado de estas operaciones lo que hace que un programa varíe su comportamiento según los datos que obtenga. Existen operaciones más sencillas o complejas, que se pueden realizar con operandos de distintos tipos de datos, como números o textos.

Ejemplos de uso de operadores

Antes de entrar a enumerar los distintos tipos de operadores vamos a ver un par de ejemplos de éstos para que nos ayuden a hacernos una idea más exacta de lo que son. En el primer ejemplo vamos a realizar una suma utilizando el operador suma.

$$3 + 5$$

Esta es una expresión muy básica que no tiene mucho sentido ella sola. Hace la suma entre los dos operandos número 3 y 5, pero no sirve de mucho porque no se hace nada con el resultado. Normalmente se combinan más de un operador para crear expresiones más útiles. La expresión siguiente es una combinación entre dos operadores, uno realiza una operación matemática y el otro sirve para guardar el resultado.

$$miVariable = 23 * 5$$

En el ejemplo anterior, el operador `*` se utiliza para realizar una multiplicación y el operador `=` se utiliza para asignar el resultado en una variable, de modo que guardemos el valor para su posterior uso.

Los operadores se pueden clasificar según el tipo de acciones que realizan. A continuación vamos a ver cada uno de estos grupos de operadores y describiremos la función de cada uno.

Operadores aritméticos

Son los utilizados para la realización de operaciones matemáticas simples como la suma, resta o multiplicación. En javascript son los siguientes:

- + Suma de dos valores
- Resta de dos valores, también puede utilizarse para cambiar el signo de un número si lo utilizamos con un solo operando -23
- * Multiplicación de dos valores
- / División de dos valores
- % El resto de la división de dos números (3%2 devolvería 1, el resto de dividir 3 entre 2)
- ++ Incremento en una unidad, se utiliza con un solo operando
- Decremento en una unidad, utilizado con un solo operando

Ejemplos

```
precio = 128 //introduzco un 128 en la variable precio
unidades = 10 //otra asignación, luego veremos operadores de asignación
factura = precio * unidades //multiplico precio por unidades, obtengo el valor
factura
resto = factura % 3 //obtengo el resto de dividir la variable factura por 3
precio++ //incrementa en una unidad el precio (ahora vale 129)
```

Operadores de asignación

Sirven para asignar valores a las variables, ya hemos utilizado en ejemplos anteriores el operador de asignación =, pero hay otros operadores de este tipo, que provienen del lenguaje C y que muchos de los lectores ya conocerán.

- = Asignación. Asigna la parte de la derecha del igual a la parte de la izquierda. A la derecha se colocan los valores finales y a la izquierda generalmente se coloca una variable donde queremos guardar el dato.
- += Asignación con suma. Realiza la suma de la parte de la derecha con la de la izquierda y guarda el resultado en la parte de la izquierda.
- = Asignación con resta
- *= Asignación de la multiplicación
- /= Asignación de la división
- %= Se obtiene el resto y se asigna

Ejemplos

```
ahorros = 7000 //asigna un 7000 a la variable ahorros
ahorros += 3500 //incrementa en 3500 la variable ahorros, ahora vale
10500
ahorros /= 2 //divide entre 2 mis ahorros, ahora quedan 5250
```

Operadores de cadenas

Las cadenas de caracteres, o variables de texto, también tienen sus propios operadores para realizar acciones típicas sobre cadenas. Aunque javascript sólo tiene un operador

para cadenas se pueden realizar otras acciones con una serie de funciones predefinidas en el lenguaje que veremos más adelante.

+ Concatena dos cadenas, pega la segunda cadena a continuación de la primera.

Ejemplo :

```
cadena1 = "hola"  
cadena2 = "mundo"  
cadenaConcatenada = cadena1 + cadena2 //cadena concatenada vale  
"holamundo"
```

Un detalle importante que se puede ver en este caso es que el operador + sirve para dos usos distintos, si sus operandos son números los suma, pero si se trata de cadenas las concatena. Esto pasa en general con todos los operadores que se repiten en el lenguaje, javascript es suficientemente listo para entender que tipo de operación realizar mediante una comprobación de los tipos que están implicados en ella.

Un caso que resultaría confuso es el uso del operador + cuando se realiza la operación con operadores texto y numéricos entremezclados. En este caso javascript asume que se desea realizar una concatenación y trata a los dos operandos como si de cadenas de caracteres se trataran, incluso si la cadena de texto que tenemos fuese un número. Esto lo veremos más fácilmente con el siguiente ejemplo.

```
miNumero = 23  
miCadena1 = "pepe"  
miCadena2 = "456"  
resultado1 = miNumero + miCadena1 //resultado1 vale "23pepe"  
resultado2 = miNumero + miCadena2 //resultado2 vale "23456"  
miCadena2 += miNumero //miCadena2 ahora vale "45623"
```

Como hemos podido ver, también en el caso del operador +=, si estamos tratando con cadenas de texto y números entremezclados, tratará a los dos operadores como si fuesen cadenas.

Operadores lógicos

Estos operadores sirven para realizar operaciones lógicas, que son aquellas que dan como resultado un verdadero o un falso, y se utilizan para tomar decisiones en nuestros scripts. En vez de trabajar con números, para realizar este tipo de operaciones se utilizan operandos booleanos, que conocimos anteriormente, que son el verdadero (true) y el falso (false). Los operadores lógicos relacionan los operandos booleanos para dar como resultado otro operando booleano, tal como podemos ver en el siguiente ejemplo.

Si tengo hambre y tengo comida entonces me pongo a comer

Nuestro programa javascript utilizaría en este ejemplo un operando booleano para tomar una decisión. Primero mirará si tengo hambre, si es cierto (true) mirará si dispongo de

comida. Si son los dos ciertos, se puede poner a comer. En caso de que no tenga comida o que no tenga hambre no comería, al igual que si no tengo hambre ni comida. El operando en cuestión es el operando Y, que valdrá verdadero (true) en caso de que los dos operandos valgan verdadero.

! Operador NO o negación. Si era true pasa a false y viceversa.

&& Operador Y, si son los dos verdaderos vale verdadero.

|| Operador O, vale verdadero si por lo menos uno de ellos es verdadero.

Ejemplo:

```
miBoleano = true
miBoleano = !miBoleano //miBoleano ahora vale false
tengoHambre = true
tengoComida = true
comoComida = tengoHambre && tengoComida
```

Operadores condicionales

Sirven para realizar expresiones condicionales todo lo complejas que deseemos. Estas expresiones se utilizan para tomar decisiones en función de la comparación de varios elementos, por ejemplo si un numero es mayor que otro o si son iguales. Los operadores condicionales se utilizan en las expresiones condicionales para tomas de decisiones. Como estas expresiones condicionales serán objeto de estudio más adelante será mejor describir los operadores condicionales más adelante. De todos modos aquí podemos ver la tabla de operadores condicionales.

== Comprueba si dos números son iguales

!= Comprueba si dos números son distintos

> Mayor que, devuelve true si el primer operador es mayor que el segundo

< Menor que, es true cuando el elemento de la izquierda es menor que el de la derecha

>= Mayor igual.

<= Menor igual

Operadores a nivel de bit

Estos son muy poco corrientes y es posible que nunca los llegues a utilizar. Su uso se realiza para efectuar operaciones con ceros y unos. Todo lo que maneja un ordenador son ceros y unos, aunque nosotros utilicemos números y letras para nuestras variables en realidad estos valores están escritos internamente en forma de ceros y unos. En algún caso podremos necesitar realizar operaciones tratando las variables como ceros y unos y para ello utilizaremos estos operandos.

& Y de bits

^ Xor de bits

| O de bits

<< >> >>> >>>= >>= <<= Varias clases de cambios

Precedencia de los operadores

La evaluación de una sentencia de las que hemos visto en los ejemplos anteriores es bastante sencilla y fácil de interpretar, pero cuando en una sentencia entran en juego multitud de operadores distintos puede haber una confusión a la hora de interpretarla y dilucidar qué operadores son los que se ejecutan antes que otros. Para marcar unas pautas en la evaluación de las sentencias y que estas se ejecuten siempre igual y con sentido común existe la precedencia de operadores, que no es más que el orden por el que se irán ejecutando las operaciones que ellos representan. En un principio todos los operadores se evalúan de izquierda a derecha, pero existen unas normas adicionales, por las que determinados operadores se evalúan antes que otros. Muchas de estas reglas de precedencia están sacadas de las matemáticas y son comunes a otros lenguajes, las podemos ver a continuación.

() [] . Paréntesis, corchetes y el operador punto que sirve para los objetos

! - ++ -- negación, negativo e incrementos

* / % Multiplicación división y módulo

+ - Suma y resta

<< >> >>> Cambios a nivel de bit

< <= > >= Operadores condicionales

== != Operadores condicionales de igualdad y desigualdad

& ^ | Lógicos a nivel de bit

&& || Lógicos booleanos

= += -= *= /= %= <<= >>= >>>= &= ^= != Asignación

En los siguientes ejemplos podemos ver cómo las expresiones podrían llegar a ser confusas, pero con la tabla de precedencia de operadores podremos entender sin errores cuál es el orden por el que se ejecutan.

$$12 * 3 + 4 - 8 / 2 \% 3$$

En este caso primero se ejecutan los operadores * / y %, de izquierda a derecha, con lo que se realizarían estas operaciones. Primero la multiplicación y luego la división por estar más a la izquierda del módulo.

$$36 + 4 - 4 \% 3$$

Ahora el módulo.

$$36 + 4 - 1$$

Por último las sumas y las restas de izquierda a derecha.

$$40 - 1$$

$$39$$

De todos modos, es importante darse cuenta que el uso de los paréntesis puede ahorrarnos muchos quebraderos de cabeza y sobretodo la necesidad de sabernos de memoria la tabla de precedencia de los operadores. Cuando veamos poco claro el orden con el que se ejecutarán las sentencias podemos utilizarlos y así forzar que se evalúe antes el trozo de expresión que se encuentra dentro de los paréntesis.

Control de tipos

Hemos visto para determinados operadores que es importante el tipo de datos que están manejando, puesto que si los datos son de un tipo se realizarán operaciones distintas que si son de otro.

Así, cuando utilizábamos el operador +, si se trataba de números los sumaba, pero si se trataba de cadenas de caracteres los concatenaba. Vemos pues que el tipo de los datos que estamos utilizando sí que importa y que tendremos que estar pendientes este detalle si queremos que nuestras operaciones se realicen tal como esperábamos.

Para comprobar el tipo de un dato se puede utilizar otro operador que está disponible a partir de javascript 1.1, el operador typeof, que devuelve una cadena de texto que describe el tipo del operador que estamos comprobando.

```
var boleano = true
var numerico = 22
var numerico_flotante = 13.56
var texto = "mi texto"
var fecha = new Date()
document.write("<br>El tipo de boleano es: " + typeof boleano)
document.write("<br>El tipo de numerico es: " + typeof numerico)
document.write("<br>El tipo de numerico_flotante es: " + typeof
numerico_flotante)
document.write("<br>El tipo de texto es: " + typeof texto)
document.write("<br>El tipo de fecha es: " + typeof fecha)
```

Este script dará como resultado lo siguiente:

```
El tipo de boleano es: boolean
El tipo de numerico es: number
El tipo de numerico_flotante es: number
El tipo de texto es: string
El tipo de fecha es: object
```

En este ejemplo podemos ver que se imprimen en la página los distintos tipos de las variables. Estos pueden ser los siguientes:

```
boolean, para los datos booleanos. (True o false)
number, para los numéricos.
string, para las cadenas de caracteres.
object, para los objetos.
```

Queremos destacar tan sólo dos detalles más:

1. Los números, ya tengan o no parte decimal, son siempre del tipo de datos numérico.
2. Una de las variables es un poco más compleja, es la variable fecha que es un objeto de la clase Date(), que se utiliza para el manejo de fechas en los scripts. La veremos más adelante, así como los objetos.

Estructuras de control

Los scripts vistos hasta ahora han sido tremendamente sencillos y lineales: se iban ejecutando las sentencias simples una detrás de la otra desde el principio hasta el fin. Sin embargo, esto no tiene porque ser siempre así, en los programas generalmente necesitaremos hacer cosas distintas dependiendo del estado de nuestras variables o realizar un mismo proceso muchas veces sin escribir la misma línea de código una y otra vez.

Para realizar cosas más complejas en nuestros scripts se utilizan las estructuras de control. Utilizándolas podemos realizar tomas de decisiones y bucles. En los siguientes capítulos vamos a conocer las distintas estructuras de control que existen en Javascript.

Toma de decisiones

Nos sirven para realizar unas acciones u otras en función del estado de las variables. Es decir, tomar decisiones para ejecutar unas instrucciones u otras dependiendo de lo que esté ocurriendo en ese instante en nuestros programas.

Por ejemplo, dependiendo si el usuario que entra en nuestra página es mayor de edad o no lo es, podemos permitirle o no ver los contenidos de nuestra página.

*Si edad es mayor que 18 entonces
Te dejo ver el contenido para adultos
Si no
Te mando fuera de la página*

En javascript podemos tomar decisiones utilizando dos enunciados distintos.

- IF
- SWITCH

Bucles

Los bucles se utilizan para realizar ciertas acciones repetidamente. Son muy utilizados a todos los niveles en la programación. Con un bucle podemos por ejemplo imprimir en una página los números del 1 al 100 sin necesidad de escribir cien veces el la instrucción imprimir.

*Desde el 1 hasta el 100
Imprimir el número actual*

En javascript existen varios tipos de bucles, cada uno está indicado para un tipo de iteración distinto y son los siguientes:

*FOR
WHILE
DO WHILE*

Estructura IF

IF es una estructura de control utilizada para tomar decisiones. Es un condicional que realiza unas u otras operaciones en función de una expresión. Funciona de la siguiente manera, primero se evalúa una expresión, si da resultado positivo se realizan las acciones relacionadas con el caso positivo.

La sintaxis de la estructura IF es la siguiente.

```
if (expresión) {  
    acciones a realizar en caso positivo  
    ...  
}
```

Opcionalmente se pueden indicar acciones a realizar en caso de que la evaluación de la sentencia de resultados negativos.

```
if (expresión) {  
    acciones a realizar en caso positivo  
    ...  
} else {  
    acciones a realizar en caso negativo  
    ...  
}
```

Las llaves engloban las acciones que se quieren realizar en caso de que se cumplan o no las expresiones. Estas llaves han de colocarse siempre, excepto en el caso de que sólo haya una instrucción como acciones a realizar, que son opcionales.

Se puede observar el sangrado (margen) que se ha colocado en cada uno de los bloques de instrucciones a ejecutar en los casos positivos y negativos. Este sangrado es totalmente opcional, sólo lo se hace hecho así para que la estructura IF se comprenda de una manera más visual. Los saltos de línea tampoco son necesarios y se han colocado también para que se vea mejor la estructura. Perfectamente se puede colocar toda la instrucción IF en la misma línea de código, pero eso no ayudará a que las cosas estén claras. Es aconsejable utilizar los sangrados y saltos de línea necesarios para que las instrucciones se puedan entender mejor, hoy y dentro de un mes, cuando te acuerdes menos de lo que hiciste en tus scripts.

Veamos algún ejemplo de condicionales IF.

```
if (dia == "lunes")  
    document.write ("Que tengas un feliz comienzo de semana")
```

Si es lunes nos deseará una feliz semana. No hará nada en caso contrario. Como en este ejemplo sólo se indica una instrucción para el caso positivo, no hace falta utilizar las llaves. Fíjese en el operador condicional que consta de dos signos igual.

Ahora otro ejemplo, un poco más largo.

```
if (credito >= precio) {  
    document.write("has comprado el artículo " + nuevoArtículo) //enseño  
    compra  
    carrito += nuevoArtículo //introduzco el artículo en el carrito de la compra  
    credito -= precio //disminuyo el crédito según el precio del artículo  
} else {  
    document.write("se te ha acabado el crédito") //informo que te falta dinero  
    window.location = "carritodelacompra.html" //voy a la página del carrito  
}
```

Este ejemplo es un poco más complejo, y también un poco ficticio. Lo que se hace es comprobar si se tiene crédito para realizar una supuesta compra. Para ello se mira si el crédito es mayor o igual que el precio del artículo, si es así se informa de la compra, se introduce el artículo en el carrito y se resta el precio al crédito acumulado. Si el precio del artículo es superior al dinero disponible se informa de la situación y se manda al navegador a la página donde se muestra el carrito de la compra.

Expresiones condicionales

La expresión a evaluar se coloca siempre entre paréntesis y está compuesta por variables que se combinan entre si mediante operadores condicionales. Recuerde que los operadores condicionales relacionaban dos variables y devuelven siempre un resultado booleano. Por ejemplo un operador condicional es el operador "es igual" (==), que devuelve true en caso de que los dos operandos sean iguales o false en caso de que sean distintos.

```
if (edad > 18)  
    document.write("puedes ver esta página para adultos")
```

En este ejemplo se utiliza el operador condicional "es mayor" (>). En este caso, devuelve true si la variable edad es mayor que 18, con lo que se ejecutará la línea siguiente que informa de que se puede ver el contenido para adultos.

Las expresiones condicionales se pueden combinar con las expresiones lógicas para crear expresiones más complejas. Recuerde que las expresiones lógicas son las que tienen como operandos a los booleanos y que devuelvan otro valor booleano. Son los operadores negación lógica, Y lógico y O lógico.

```
if (bateria == 0 && redElectrica = 0)
    document.write("tu ordenador portatil se va a apagar en segundos")
```

Lo que se hace es comprobar si la batería del supuesto ordenador está a cero (acabada) y también se comprueba si el ordenador no tiene red eléctrica (está desenchufado). Luego, el operador lógico los relaciona con un Y, de modo que si está sin batería Y sin red eléctrica, se informa que el ordenador se va a apagar.

Sentencias IF anidadas

Para hacer estructuras condicionales más complejas podemos anidar sentencias IF, es decir, colocar estructuras IF dentro de otras estructuras IF. Con un solo IF podemos evaluar y realizar una acción u otra según dos posibilidades, pero si tenemos más posibilidades que evaluar debemos anidar ifs para crear el flujo de código necesario para decidir correctamente.

Por ejemplo, si se desea comprobar si un número es mayor menor o igual que otro, se tiene que evaluar tres posibilidades distintas. Primero se puede comprobar si los dos números son iguales, si lo son, ya se ha resuelto el problema, pero si no son iguales todavía se tiene que ver cuál de los dos es mayor. Veamos este ejemplo en código Javascript.

```
var numero1=23
var numero2=63
if (numero1 == numero2){
    document.write("Los dos números son iguales")
else {
    if (numero1 > numero2) {
        document.write("El primer número es mayor que el segundo")
    }else{
        document.write("El primer número es menor que el segundo")
    }
}
}
```

El flujo del programa es como se comentó antes, primero se evalúa si los dos números son iguales. En caso positivo se muestra un mensaje informándolo. En caso contrario ya se sabe que son distintos, pero aun se debe averiguar cuál de los dos es mayor. Para eso se hace otra comparación para saber si el primero es mayor que el segundo. Si esta comparación da resultados positivos mostramos un mensaje diciendo que el primero es mayor que el segundo, en caso contrario se indica que el primero es menor que el segundo.

Recuerde que las llaves son en este caso opcionales, pues sólo se ejecuta una sentencia para cada caso. Además, los saltos de línea y los sangrados también opcionales en todo caso y nos sirven sólo para ver el código de una manera más ordenada. Mantener el código bien estructurado y escrito de una manera comprensible es muy importante, ya que hace la vida más agradable a la hora de programar y más adelante cuando se tenga que revisar los programas.

Operador IF

Hay un operador que no se ha visto todavía y es una forma más esquemática de realizar algunos IF sencillos. Proviene del lenguaje C, donde se escriben muy pocas líneas de código que resulta muy elegante. Este operador es un claro ejemplo de ahorro de líneas y caracteres al escribir los scripts. Se verá rápidamente, pues la única razón por la que se incluye es para que se sepa que existe y si se encuentra en alguna ocasión por ahí sepa identificarlo y cómo funciona.

Un ejemplo de uso del operador IF se puede ver a continuación.

Variable = (condición) ? valor1 : valor2

Este ejemplo no sólo realiza una comparación de valores, además asigna un valor a una variable. Lo que hace es evaluar la condición (colocada entre paréntesis) y si es positiva asigna el valor1 a la variable y en caso contrario le asigna el valor2. Veamos un ejemplo:

momento = (hora_actual < 12) ? "Antes del mediodía" : "Después del mediodía"

Este ejemplo mira si la hora actual es mayor que 12. Si es así, es que ahora es antes del mediodía, así que asigna "Antes del mediodía" a la variable momento. Si la hora es mayor o igual a 12 es que ya es después de mediodía, con lo que se asigna el texto "Después del mediodía" a la variable momento.

Estructura SWITCH

Es la otra expresión disponible en Javascript para tomar decisiones en función de distintos estados de las variables. Esta expresión se utiliza cuando se tienen múltiples posibilidades como resultado de la evaluación de una sentencia.

La estructura SWITCH se incorporó a partir de la versión 1.2 de Javascript (Netscape 4 e Internet Explorer 4). Su sintaxis es la siguiente.

```
switch (expersión) {  
  case valor1:  
    Sentencias a ejecutar si la expresión tiene como valor a valor1  
    break  
  case valor2:  
    Sentencias a ejecutar si la expresión tiene como valor a valor2  
    break  
  case valor3:  
    Sentencias a ejecutar si la expresión tiene como valor a valor3  
    break  
  default:  
    Sentencias a ejecutar si el valor no es ninguno de los anteriores  
}
```

La expresión se evalúa, si vale valor1 se ejecutan las sentencias relacionadas con ese caso. Si la expresión vale valor2 se ejecutan las instrucciones relacionadas con ese valor y así sucesivamente, por tantas opciones como se desee. Finalmente, para todos los casos no contemplados anteriormente se ejecuta el caso por defecto.

La palabra `break` es opcional, pero si no se pone a partir de que se encuentre coincidencia con un valor se ejecutarán todas las sentencias relacionadas con este y todas las siguientes. Es decir, si en el esquema anterior no hubiese ningún `break` y la expresión valiese valor1, se ejecutarían las sentencias relacionadas con valor1 y también las relacionadas con valor2, valor3 y `default`.

También es opcional la opción `default` u opción por defecto.

Veamos un ejemplo de uso de esta estructura. Supongamos que queremos indicar que día de la semana es. Si el día es 1 (lunes) sacar un mensaje indicándolo, si el día es 2 (martes) se debe sacar un mensaje distinto y así sucesivamente para cada día de la semana, menos en el 6 (sábado) y 7 (domingo) que queremos mostrar el mensaje "es fin de semana". Para días mayores que 7 indicaremos que ese día no existe.

```
Switch (dia_de_la_semana) {
  case 1:
    document.write("Es Lunes")
    break
  case 2:
    document.write("Es Martes")
    break
  case 3:
    document.write("Es Miércoles")
    break
  case 4:
    document.write("Es Jueves")
    break
  case 5:
    document.write("Es viernes")
    break
  case 6:
  case 7:
    document.write("Es fin de semana")
    break
  default:
    document.write("Ese día no existe")
}
```

El ejemplo es relativamente sencillo, solamente puede tener una pequeña dificultad, consistente en interpretar lo que pasa en el caso 6 y 7, que se había dicho que se debía que mostrar el mismo mensaje. En el caso 6 en realidad no se indica ninguna instrucción, pero como tampoco se coloca un `break` se ejecuta la sentencia o sentencias del caso siguiente, que corresponden con la sentencia indicada en el caso 7 que es el mensaje que

informa que es fin de semana. Si el caso es 7 simplemente se indica que es fin de semana, tal como se pretendía.

Bucle FOR

El bucle FOR se utiliza para repetir mas instrucciones un determinado número de veces. De entre todos los bucles, el FOR se suele utilizar cuando sabemos seguro el número de veces que queremos que se ejecute la sentencia. La sintaxis del bucle se muestra a continuación.

```
for (inicialización;condición;actualización) {  
    sentencias a ejecutar en cada iteración  
}
```

El bucle FOR tiene tres partes incluidas entre los paréntesis. La primera es la inicialización, que se ejecuta solamente al comenzar la primera iteración del bucle. En esta parte se suele colocar la variable que utilizaremos para llevar la cuenta de las veces que se ejecuta el bucle.

La segunda parte es la condición, que se evaluará cada vez que comience la iteración del bucle. Contiene una expresión para comprobar cuándo se ha de detener el bucle, o mejor dicho, la condición que se debe cumplir para que continúe la ejecución del bucle.

Por último se tiene la actualización, que sirve para indicar los cambios que se quieren ejecutar en las variables cada vez que termina la iteración del bucle, antes de comprobar si se debe seguir ejecutando.

Después del for se colocan las sentencias que se quieren ejecutar en cada iteración, acotadas entre llaves.

Un ejemplo de utilización de este bucle se puede ver a continuación, donde se imprimirán los números del 0 al 10.

```
var i  
for (i=0;i<=10;i++) {  
    document.write(i)  
}
```

En este caso se inicializa la variable i a 0. Como condición para realizar una iteración, se tiene que cumplir que la variable i sea menor o igual que 10. Como actualización se incrementará en 1 la variable i.

Como se puede comprobar, este bucle es muy potente, ya que en una sola línea podemos indicar muchas cosas distintas y muy variadas.

Por ejemplo si se quiere escribir los número del 1 al 1.000 de dos en dos se escribirá el siguiente bucle.

```
for (i=1;i<=1000;i+=2)
    document.write(i)
```

En cada iteración actualizamos el valor de i incrementándolo en 2 unidades.

Otro detalle, no se utilizan las llaves englobando las instrucciones del bucle FOR porque sólo tiene una sentencia y en este caso no es obligado, tal como pasaba con las instrucciones del IF.

Si se quiere contar descendentemente del 343 al 10 utilizamos este bucle.

```
for (i=343;i>=10;i--)
    document.write(i)
}
```

En este caso se decrementa en una unidad la variable i en cada iteración.

Ejemplo

Se trata de hacer un bucle que escriba en una página web los encabezamientos desde <H1> hasta <H6> con un texto que ponga "Encabezado de nivel x".

Lo que se desea escribir en una página web mediante Javascript es lo siguiente:

```
<H1>Encabezado de nivel 1</H1>
<H2>Encabezado de nivel 2</H2>
<H3>Encabezado de nivel 3</H3>
<H4>Encabezado de nivel 4</H4>
<H5>Encabezado de nivel 5</H5>
<H6>Encabezado de nivel 6</H6>
```

Para ello se tiene que hacer un bucle que empiece en 1 y termine en 6 y en cada iteración se escribe el encabezado que toca.

```
for (i=1;i<=6;i++) {
    document.write("<H" + i + ">Encabezado de nivel " + i + "</H" + i + ">")
}
```

Bucles WHILE y DO WHILE

Veamos ahora los dos tipos de bucles WHILE que podemos utilizar en Javascript y los usos de cada uno.

- **Bucle WHILE**

Estos bucles se utilizan cuando queremos repetir la ejecución de unas sentencias un número indefinido de veces, siempre que se cumpla una condición. Se más sencillo de comprender que el bucle FOR, pues 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.

```
while (condición){  
    sentencias a ejecutar  
}
```

Un ejemplo de código donde se utiliza este bucle se puede ver a continuación.

```
var color = ""  
while (color != "rojo")  
    color = dame un color  
}
```

Este es un ejemplo de lo más sencillo que se puede hacer con un bucle while. Lo que hace es pedir que el usuario introduzca un color mientras que el color no sea rojo. Para ejecutar un bucle como este primero tenemos que inicializar la variable que vamos utilizar en la condición de iteración del bucle. Con la variable inicializada podemos escribir el bucle, que comprobará para ejecutarse que el la variable color sea distinto de "rojo". En cada iteración del bucle se pide un nuevo color al usuario para actualizar la variable color y se termina la iteración, con lo que retornamos al principio del bucle, donde tenemos que volver a evaluar si lo que hay en la variable color es "rojo" y así sucesivamente mientras que no se haya introducido como color el texto "rojo". Obviamente la expresión dame un color no es Javascript, pero como no sabemos todavía cómo escribir eso en Javascript es mejor verlo más adelante.

- **Bucle DO...WHILE**

Es el último de los bucles que hay en Javascript. Se utiliza generalmente cuando no sabemos cuantas veces se habrá de ejecutar el bucle, igual que el bucle WHILE, con la diferencia de que sabemos seguro que el bucle por lo menos se ejecutará una vez.

Este tipo de bucle se introdujo en Javascript 1.2, por lo que no todos los navegadores los soportan, sólo los de versión 4 o superior. En cualquier caso, cualquier código que quieras escribir con DO...WHILE se puede escribir también utilizando un bucle WHILE, con lo que en navegadores antiguos deberás traducir tu bucle DO...WHILE por un bucle WHILE.

La sintaxis es la siguiente.

```
do {  
    sentencias del bucle  
} while (condición)
```

El bucle se ejecuta siempre una vez y al final se evalúa la condición para decir si se ejecuta otra vez el bucle o se termina su ejecución.

Veamos el ejemplo que escribimos para un bucle WHILE en este otro tipo de bucle.

```
var color
do {
    color = dame un color
} while (color != "rojo")
```

Este ejemplo funciona exactamente igual que el anterior, excepto que no tuvimos que inicializar la variable color antes de introducirnos en el bucle. Pide un color mientras que el color introducido es distinto que "rojo".

Ejemplo

Vamos a ver a continuación un ejemplo más práctico sobre cómo trabajar con un bucle WHILE. Como resulta muy difícil hacer ejemplos prácticos con lo poco que sabemos sobre Javascript, vamos a adelantar una instrucción que aun no conocemos.

En este ejemplo vamos a declarar una variable e inicializarla a 0. Luego iremos sumando a esa variable un número aleatorio del 1 al 100 hasta que sumemos 1.000 o más, imprimiendo el valor de la variable suma después de cada operación. Será necesario utilizar el bucle WHILE porque no sabemos exactamente el número de iteraciones que tendremos que realizar.

```
var suma = 0
while (suma < 1000){
    suma += parseInt(Math.random() * 100)
    document.write (suma + "<br>")
}
```

Suponemos que por lo que respecta al bucle WHILE no habrá problemas, pero donde si que puede haberlos es en la sentencia utilizada para tomar un número aleatorio. Sin embargo, no es necesario explicar aquí la sentencia porque lo tenemos planeado hacer más adelante.

Break y continue

De manera adicional al uso de las distintas estructuras de bucle se pueden utilizar dos instrucciones para

- Detener la ejecución de un bucle y salirse de él
- Detener la iteración actual y volver al principio del bucle.

Son las instrucciones break y continue.

- **Break**

Se detiene un bucle utilizando la palabra break. Detener un bucle significa salirse de él y dejarlo todo como está para continuar con el flujo del programa inmediatamente después del bucle.

```
for (i=0;i<10;i++){  
    document.write (i)  
    escribe = dime si continúo  
    if (escribe == "no")  
        break  
}
```

Este ejemplo escribe los números del 0 al 9 y en cada iteración del bucle pregunta al usuario si desea continuar. Si el usuario dice cualquier cosa continua excepto cuando dice "no" que entonces se sale del bucle y deja la cuenta por donde se había quedado.

- **Continue**

Sirve para volver al principio del bucle en cualquier momento, sin ejecutar las líneas que haya por debajo de la palabra continue.

```
var i=0  
while (i<7){  
    incrementar = dime si incremento  
    if (incrementar == "no")  
        continue  
    i++  
}
```

Este ejemplo, en condiciones normales contaría hasta desde i=0 hasta i=7, pero cada vez que se ejecuta el bucle pregunta al usuario si desea incrementar la variable o no. Si introduce "no" se ejecuta la sentencia continue, con lo que se vuelve al principio del bucle sin llegar a incrementar en 1 la variable i, ya que se ignoran las sentencias que hayan por debajo del continue.

Ejemplo

Un ejemplo más práctico sobre estas instrucciones se puede ver a continuación. Se trata de un bucle FOR planeado para llegar hasta 1.000 pero que lo vamos a parar con break cuando llegemos a 333.

```
for (i=0;i<=1000;i++){  
    document.write(i + "<br>")  
    if (i==333)  
        break;  
}
```

Bucles anidados en Javascript

Anidar un bucle consiste en meter ese bucle dentro de otro. La anidación de bucles es necesaria para hacer determinados procesamientos un poco más complejos que los que se han visto en los ejemplos anteriores.

Un bucle anidado tiene una estructura como la que sigue:

```
for (i=0;i<10;i++){  
  for (j=0;j<10;j++) {  
    document.write(i + "-" + j)  
  }  
}
```

La ejecución funcionará de la siguiente manera. Para empezar se inicializa el primer bucle, con lo que la variable *i* valdrá 0 y a continuación se inicializa el segundo bucle, con lo que la variable *j* valdrá también 0. En cada iteración se imprime el valor de la variable *i*, un guión ("-") y el valor de la variable *j*, como las dos variables valen 0, se imprimirá el texto "0-0" en la página web.

El bucle que está anidado (más hacia dentro) es el que más veces se ejecuta, en este ejemplo, para cada iteración del bucle más externo el bucle anidado se ejecutará por completo una vez, es decir, hará sus 10 iteraciones. En la página web se escribirían estos valores, en la primera iteración del bucle externo y desde el principio:

```
0-0  
0-1  
0-2  
0-3  
0-4  
0-5  
0-6  
0-7  
0-8  
0-9
```

Para cada iteración del bucle externo se ejecutarán las 10 iteraciones del bucle interno o anidado. Se ha visto la primera iteración, ahora vamos a ver las siguientes iteraciones del bucle externo. En cada una acumula una unidad en la variable *i*, con lo que saldrían estos valores.

```
1-0  
1-1  
1-2  
1-3  
1-4  
1-5  
1-6
```

1-7
1-8
1-9

Y luego estos.

2-0
2-1
2-2
2-3
2-4
2-5
2-6
2-7
2-8
2-9

Así hasta que se terminen los dos bucles, que sería cuando se alcanzase el valor 9-9.

Veamos un ejemplo muy parecido al anterior, aunque un poco más útil. Se trata de imprimir en la página las todas las tablas de multiplicar. Del 1 al 9, es decir, la tabla del 1, la del 2, del 3...

```
for (i=1;i<10;i++){  
  document.write("<br><b>La tabla del " + i + ":</b><br>")  
  for (j=1;j<10;j++) {  
    document.write(i + " x " + j + ": ")  
    document.write(i*j)  
    document.write("<br>")  
  }  
}
```

Con el primer bucle se controla la tabla actual y con el segundo bucle se desarrolla. En el primer bucle se escribe una cabecera, en negrita, indicando la tabla que se está escribiendo, primero la del 1 y luego las demás en orden ascendente hasta el 9. Con el segundo bucle se escribe cada uno de los valores de cada tabla.

Funciones en Javascript

Ahora vamos a ver un tema muy importante, sobretodo para los que no han programado nunca y con Javascript están dando sus primeros pasos en el mundo de la programación ya que se verá un concepto nuevo, el de función, y los usos que tiene. Para los que ya conozcan el concepto de función también será útil, pues también se verá la sintaxis y funcionamiento de las funciones en Javascript.

¿Qué es una función?

A la hora de hacer un programa ligeramente grande existen determinados procesos que se pueden concebir de forma independiente, y que son más sencillos de resolver que el problema entero. Además, estos suelen ser realizados repetidas veces a lo largo de la ejecución del programa. Estos procesos se pueden agrupar en una función, definida para que no se etenga que repetir una y otra vez ese código en nuestros scripts, sino que simplemente llamamos a la función y ella se encarga de hacer todo lo que debe.

Así que podemos ver una función como una serie de instrucciones que englobamos dentro de un mismo proceso. Este proceso se podrá luego ejecutar desde cualquier otro sitio con solo llamarlo. Por ejemplo, en una página web puede haber una función para cambiar el color del fondo y desde cualquier punto de la página podríamos llamarla para que nos cambie el color cuando lo deseemos.

Las funciones se utilizan constantemente, no sólo las propias, sino también las que ya están definidas en el sistema, pues todos los lenguajes de programación tienen un montón de funciones para realizar procesos habituales como por ejemplo obtener la hora, imprimir un mensaje en la pantalla o convertir variables de un tipo a otro. Ya se ha visto alguna función en nuestros sencillos ejemplos anteriores cuando se hacía un `document.write()` en realidad se estaba llamando a la función `write()` asociada al documento de la página que escribe un texto en la página. Vamos primero a ver cómo realizar nuestras propias funciones y cómo llamarlas luego.

Cómo se escribe una función

Una función se debe definir con una sintaxis especial que vamos a conocer a continuación.

```
function nombrefuncion (){  
    instrucciones de la función  
    ...  
}
```

Primero se escribe la palabra `function`, reservada para este uso. Seguidamente se escribe el nombre de la función, que como los nombres de variables puede tener números, letras y algún carácter adicional como en guión bajo. A continuación se colocan entre llaves las distintas instrucciones de la función. Las llaves en el caso de las funciones no son opcionales, además es útil colocarlas siempre como se ve en el ejemplo, para que se vea fácilmente la estructura de instrucciones que engloba la función.

Veamos un ejemplo de función para escribir en la página un mensaje de bienvenida dentro de etiquetas `<H1>` para que quede más resaltado.

```
function escribirBienvenida(){  
    document.write("<H1>Hola a todos</H1>")  
}
```

Simplemente escribe en la página un texto, es una función tan sencilla que el ejemplo no expresa suficientemente el concepto de función, pero ya se verán otras más complejas. Las etiquetas H1 no se escriben en la página, sino que son interpretadas como el significado de la misma, en este caso que se escribe un encabezado de nivel 1. Como se está escribiendo en una página web, al poner etiquetas HTML se interpretan como lo que son.

¿Cómo llamar a una función?

Para ejecutar una función se tiene que llamar en cualquier parte de la página, con eso se consigue que se ejecuten todas las instrucciones que tiene la función entre las dos llaves. Para ejecutar la función utilizamos su nombre seguido de los paréntesis.

```
NombreDeLaFuncion()
```

¿Dónde colocamos las funciones?

En principio, podemos colocar las funciones en cualquier parte de la página, siempre entre etiquetas <SCRIPT>, claro está. No obstante existe una limitación a la hora de colocarla con relación a los lugares desde donde se la llame. Lo más normal es colocar la función antes de cualquier llamada a la misma y así seguro que nunca nos equivocaremos.

En concreto, la función se debe definir en el bloque <SCRIPT> donde esté la llamada a la función, aunque es indiferente si la llamada se encuentra antes o después la función, dentro del mismo bloque <SCRIPT>.

```
<SCRIPT>
miFuncion()
function miFuncion(){
    //hago algo...
    document.write("Esto va bien")
}
</SCRIPT>
```

Este ejemplo funciona correctamente porque la función está declarada en el mismo bloque que su llamada.

También es válido que la función se encuentre en un bloque <SCRIPT> anterior al bloque donde está la llamada.

```
<HTML>
<HEAD>
  <TITLE>MI PÁGINA</TITLE>
<SCRIPT>
function miFuncion(){
    //hago algo...
```

```
        document.write("Esto va bien")
    }
</SCRIPT>
</HEAD>
<BODY>

<SCRIPT>
miFuncion()
</SCRIPT>

</BODY>
</HTML>
```

Vemos un código completo sobre cómo podría ser una página web donde las funciones están en la cabecera. Un lugar muy bueno donde colocarlas, porque se supone que en la cabecera no se van a utilizar todavía y siempre podremos disfrutar de ellas en el cuerpo porque ya han sido declaradas seguro.

Esto último en cambio sería un error.

Lo que será un error es una llamada a una función que se encuentra declarada en un bloque <SCRIPT> posterior.

```
<SCRIPT>
miFuncion()
</SCRIPT>

<SCRIPT>
function miFuncion(){
    //hago algo...
    document.write("Esto va bien")
}
</SCRIPT>
```

Parámetros de las funciones

Las estructuras que se han visto anteriormente sobre funciones no son las únicas que debemos aprender para manejarlas en toda su potencia. Las funciones también tienen una entrada y una salida, que se pueden utilizar para recibir y devolver datos.

Parámetros

Los parámetros se usan para mandar valores a la función, con los que ella trabajará para realizar las acciones. Son los valores de entrada que recibe una función. Por ejemplo, una función que realizase una suma de dos números tendría como parámetros a esos dos números. Los dos números son la entrada, así como la salida sería el resultado, pero eso lo veremos más tarde.

Veamos un ejemplo anterior en el que se creaba una función para mostrar un mensaje de bienvenida en la página web, pero al que ahora le vamos a pasar un parámetro que contendrá el nombre de la persona a la que hay que saludar.

```
function escribirBienvenida(nombre){
    document.write("<H1>Hola " + nombre + "</H1>")
}
```

Como se puede ver en el ejemplo, para definir en la función un parámetro tenemos que poner el nombre de la variable que va a almacenar el dato que le pasemos. Esa variable, que en este caso se llama nombre, tendrá como valor el dato que le pasemos a la función cuando la llamemos, además, la variable tendrá vida durante la ejecución de la función y dejará de existir cuando la función termine su ejecución.

Para llamar a una función que tiene parámetros se coloca entre paréntesis el valor del parámetro. Para llamar a la función del ejemplo hay que escribir:

```
escribirBienvenida("Alberto García")
```

Al llamar a la función así, el parámetro nombre toma como valor "Alberto García" y al escribir el saludo por pantalla escribirá "Hola Alberto García" entre etiquetas <H1>.

Los parámetros pueden recibir cualquier tipo de datos, numérico, textual, booleano o un objeto. Realmente no especificamos el tipo del parámetro, por eso debemos tener un cuidado especial al definir las acciones que se realizan dentro de la función y al pasarle valores a la función para asegurarnos que todo es consecuente con los tipos de nuestras variables o parámetros.

Múltiples parámetros

Una función puede recibir tantos parámetros como queramos y para expresarlo se colocan los parámetros separados por comas dentro de los paréntesis. Veamos rápidamente la sintaxis para que la función de antes reciba dos parámetros, el primero el nombre al que saludar y el segundo el color del texto.

```
function escribirBienvenida(nombre,colorTexto){
    document.write("<FONT color=" + colorTexto + ">")
    document.write("<H1>Hola " + nombre + "</H1>")
    document.write("</FONT>")
}
```

Llamamos a la función con esta sintaxis. Entre los paréntesis colocaremos los valores de los parámetros.

```
var miNombre = "Pepe"
var miColor = "red"
escribirBienvenida(miNombre,miColor)
```


Se ha colocado entre los paréntesis dos variables en lugar de dos textos entrecorillados. Cuando colocamos variables entre los parámetros en realidad lo que estamos pasando a la función son los valores que contienen las variables y no las mismas variables.

Parámetros se pasan por valor

Al hilo del uso de parámetros en nuestros programas Javascript tenemos que indicar que los parámetros de las funciones se pasan por valor. Esto quiere decir que aunque modifiquemos un parámetro en una función la variable original que habíamos pasado no cambiará su valor. Se puede ver fácilmente con un ejemplo.

```
function pasoPorValor(miParametro){
    miParametro = 32
    document.write("he cambiado el valor a 32")
}
var miVariable = 5
pasoPorValor(miVariable)
document.write ("el valor de la variable es: " + miVariable)
```

En el ejemplo se tiene una función que recibe un parámetro y que modifica el valor del parámetro asignándole el valor 32. También se tiene una variable, que inicializamos a 5 y posteriormente llamamos a la función pasándole esta variable como parámetro. Como dentro de la función modificamos el valor del parámetro podría pasar que la variable original cambiase de valor, pero como los parámetros no modifican el valor original de las variables esta no cambia de valor. De este modo, al imprimir en pantalla el valor de miVariable se imprimirá el número 5, que es el valor original de la variable, en lugar de 32 que era el valor con el que habíamos actualizado el parámetro.

En javascript sólo se pueden pasar las variables por valor.

Valores de retorno

Las funciones también pueden retornar valores, de modo que al ejecutar la función se podrá realizar acciones y dar un valor como salida. Por ejemplo, una función que calcula el cuadrado de un número tendrá como entrada -tal como vimos- a ese número y como salida tendrá el valor resultante de hallar el cuadrado de ese número. Una función que devuelva el día de la semana tendría como salida en día de la semana.

Veamos un ejemplo de función que calcula la media de dos números. La función recibirá los dos números y retornará el valor de la media.

```
function media(valor1,valor2){
    var resultado
    resultado = (valor1 + valor2) / 2
    return resultado
}
```

Para especificar el valor que retornará la función se utiliza la palabra `return` seguida del valor que se desea devolver. En este caso se devuelve el contenido de la variable `resultado`, que contiene la media de los dos números.

Para recibir los valores que devuelve una función se coloca el operador de asignación `=`. Para ilustrar esto se puede ver este ejemplo, que llamará a la función `media()` y guardará el resultado de la media en una variable para luego imprimirla en la página.

```
var miMedia
miMedia = media(12,8)
document.write (miMedia)
```

Múltiples return

En una misma función podemos colocar más de un `return`. Lógicamente sólo vamos a poder retornar una cosa, pero dependiendo de lo que haya sucedido en la función podrá ser de un tipo u otro, con unos datos u otros.

En esta función podemos ver un ejemplo de utilización de múltiples `return`. Se trata de una función que devuelve un 0 si el parámetro recibido era par y el valor del parámetro si este era impar.

```
function multipleReturn(numero){
  var resto = numero % 2
  if (resto == 0)
    return 0
  else
    return numero
}
```

Para averiguar si un número es par hallamos el resto de la división al dividirlo entre 2. Si el resto es cero es que era par y devolvemos un 0, en caso contrario -el número es impar- devolvemos el parámetro recibido.

Ámbito de las variables en funciones

Dentro de las funciones podemos declarar variables, incluso los parámetros son como variables que se declaran en la cabecera de la función y que se inicializan al llamar a la función. Todas las variables declaradas en una función son locales a esa función, es decir, solo tendrán validez durante la ejecución de la función.

Podemos declarar variables en funciones que tengan el mismo nombre que una variable global a la página. Entonces, dentro de la función la variable que tendrá validez es la variable local y fuera de la función tendrá validez la variable global a la página.

En cambio, si no declaramos las variables en las funciones se entenderá por javascript que estamos haciendo referencia a una variable global a la página, de modo que si no

está creada la variable la crea, pero siempre global a la página en lugar de local a la función.

Arrays en Javascript

En los lenguajes de programación existen estructuras de datos especiales que nos sirven para guardar información más compleja que simples variables. Una estructura típica en todos los lenguajes es el Array, que es como una variable donde podemos introducir varios valores, en lugar de solamente uno como ocurre con la variables normales.

Los arrays nos permiten guardar varias variables y acceder a ellas de manera independiente, es como tener una variable con distintos compartimentos donde podemos introducir datos distintos. Para ello utilizamos un índice que nos permite especificar el compartimiento o posición a la que nos estamos refiriendo.

Los arrays se introdujeron en versiones Javascript 1.1 o superiores, es decir, solo los podemos utilizar a partir de los navegadores 3.0. Para navegadores antiguos se puede simular el array utilizando sintaxis de programación orientada a objetos.

Creación de Arrays

El primer paso para utilizar un array es crearlo. Para ello utilizamos un objeto Javascript ya implementado en el navegador. Veremos en adelante un tema para explicar lo que es la orientación a objetos, aunque no será necesario para poder entender el uso de los arrays. Esta es la sentencia para crear un objeto array:

```
var miArray = new Array()
```

Esto crea un array en la página que esta ejecutándose. El array se crea sin ningún contenido, es decir, no tendrá ninguna casilla o compartimiento creado. También podemos crear el array especificando el número de compartimentos que va a tener.

```
var miArray = new Array(10)
```

En este caso indicamos que el array va a tener 10 posiciones, es decir, 10 casillas donde guardar datos.

Es importante que nos fijemos que la palabra Array en código Javascript se escribe con la primera letra en mayúscula. Como en Javascript las mayúsculas y minúsculas si que importan, si lo escribimos en minúscula no funcionará.

Tanto se indique o no el número de casillas del array, podemos introducir en el array cualquier dato. Si la casilla está creada se introduce simplemente y si la casilla no estaba creada se crea y luego se introduce el dato, con lo que el resultado final es el mismo. Esta creación de casillas es dinámica y se produce al mismo tiempo que los scripts se ejecutan. Veamos a continuación cómo introducir valores en nuestros arrays.

```
miArray[0] = 290
```

```
miArray[1] = 97  
miArray[2] = 127
```

Se introducen indicando entre corchetes el índice de la posición donde queríamos guardar el dato. En este caso introducimos 290 en la posición 0, 97 en la posición 1 y 127 en la 2. Los arrays empiezan siempre en la posición 0, así que un array que tenga por ejemplo 10 posiciones, tendrá casillas de la 0 a la 9. Para recoger datos de un array lo hacemos igual: poniendo entre corchetes el índice de la posición a la que queremos acceder. Veamos cómo se imprimiría en la pantalla el contenido de un array.

```
var miArray = new Array(3)  
  
miArray[0] = 155  
miArray[1] = 4  
miArray[2] = 499  
  
for (i=0;i<3;i++){  
    document.write("Posición " + i + " del array: " + miArray[i])  
    document.write("<br>")  
}
```

Se ha creado un array con tres posiciones, luego se ha introducido un valor en cada una de las posiciones del array y finalmente las se ha impreso. En general, el recorrido por arrays para imprimir sus posiciones o cualquier otra cosa se hace utilizando bucles. En este caso utilizamos un bucle FOR que va desde el 0 hasta el 2.

Tipos de datos en los arrays

En las casillas de los arrays podemos guardar datos de cualquier tipo. Se puede ver un array donde se introducen datos de tipo carácter.

```
miArray[0] = "Hola"  
miArray[1] = "a"  
miArray[2] = "todos"
```

Incluso, en Javascript se pueden guardar distintos tipos de datos en las casillas de un mismo array. Es decir, se pueden introducir números en unas casillas, textos en otras, booleanos o cualquier otra cosa que se desee.

```
miArray[0] = "desarrolloweb.com"  
miArray[1] = 1275  
miArray[1] = 0.78  
miArray[2] = true
```

Longitud de los arrays

Todos los arrays en javascript, aparte de almacenar el valor de cada una de sus posiciones también almacenan el número de posiciones que tienen. Para ello utilizan una

propiedad del objeto array, la propiedad length. Ya veremos qué es una propiedad, pero para nuestro caso podemos imaginarnos que es como una variable, adicional a las posiciones, que almacena un número igual al número de casillas del array.

Para acceder a una propiedad de un objeto se ha de utilizar el operador punto. Se escribe el nombre del array que queremos acceder al número de posiciones que tiene, sin corchetes ni paréntesis, seguido de un punto y la palabra length.

```
var miArray = new Array()

miArray[0] = 155
miArray[1] = 499
miArray[2] = 65

document.write("Longitud del array: " + miArray.length)
```

Este código imprimiría en pantalla el número de posiciones del array, que en este caso es 3. Recuerde que un array con 3 posiciones abarca desde la posición 0 a la 2.

Es muy habitual que se utilice la propiedad length para poder recorrer un array por todas sus posiciones. Para ilustrarlo vamos a ver un ejemplo de recorrido por este array para mostrar sus valores.

```
for (i=0;i<miArray.length;i++){
    document.write(miArray[i])
}
```

Hay que fijarse que el bucle for se ejecuta siempre que i valga menos que la longitud del array, extraída de su propiedad length.

El siguiente ejemplo nos sirve para conocer mejor los recorridos por los arrays, el funcionamiento de la propiedad length y la creación dinámica de nuevas posiciones. Vamos a crear un array con 2 posiciones y rellenar su valor. Posteriormente se introduce un valor en la posición 5 del array. Finalmente imprimiremos todas las posiciones del array para ver lo que pasa.

```
var miArray = new Array(2)

miArray[0] = "Colombia"
miArray[1] = "Estados Unidos"

miArray[5] = "Brasil"

for (i=0;i<miArray.length;i++){
    document.write("Posición " + i + " del array: " + miArray[i])
    document.write("<br>")
}
```

El ejemplo es sencillo. Se puede apreciar que se hace un recorrido por el array desde 0 hasta el número de posiciones del array (indicado por la propiedad length). En el recorrido se va imprimiendo el número de la posición seguido del contenido del array en esa posición. Pero se puede tener una duda al preguntarnos cuál será el número de elementos de este array, ya que lo habíamos declarado con 2 y luego le se ha introducido un tercero en la posición 5. Al ver la salida del programa podremos contestar nuestras preguntas. Será algo parecido a esto:

```
Posición 0 del array: Colombia  
Posición 1 del array: Estados Unidos  
Posición 2 del array: null  
Posición 3 del array: null  
Posición 4 del array: null  
Posición 5 del array: Brasil
```

Se puede ver claramente que el número de posiciones es 6, de la 0 a la 5. Lo que ha ocurrido es que al introducir un dato en la posición 5, todas las casillas que no estaban creadas hasta la quinta se crean también.

Las posiciones de la 2 a la 4 están sin inicializar. En este caso nuestro navegador ha escrito la palabra null para expresar esto, pero otros navegadores podrán utilizar la palabra undefined. Ya se verá más adelante qué es este null y dónde se puede utilizar, lo importante ahora es comprender cómo trabajan los arrays y correcta utilización.

Arrays multidimensionales

Los arrays multidimensionales son un estructuras de datos que almacenan los valores en más de una dimensión. Los arrays que se han visto hasta ahora almacenan valores en una dimensión, por eso para acceder a las posiciones utilizamos tan solo un índice. Los arrays de 2 dimensiones guardan sus valores, por decirlo de alguna manera, en filas y columnas y por ello necesitaremos dos índices para acceder a cada una de sus posiciones.

Dicho de otro modo, un array multidimensional es como un contenedor que guardara más valores para cada posición, es decir, como si los elementos del array fueran a su vez otros arrays.

En Javascript no existe un auténtico objeto array-multidimensional. Para utilizar estas estructuras podremos definir arrays que donde en cada una de sus posiciones habrá otro array. En nuestros programas podremos utilizar arrays de cualquier dimensión, veremos a continuación cómo trabajar con arrays de dos dimensiones, que serán los más comunes.

En este ejemplo se va a crear un array de dos dimensiones donde se tendrá por un lado ciudades y por el otro la temperatura media que hace en cada una durante de los meses de invierno.

```
var temperaturas_medias_ciudad0 = new Array(3)  
temperaturas_medias_ciudad0[0] = 12
```

```

temperaturas_medias_ciudad0[1] = 10
temperaturas_medias_ciudad0[2] = 11

var temperaturas_medias_ciudad1 = new Array (3)
temperaturas_medias_ciudad1[0] = 5
temperaturas_medias_ciudad1[1] = 0
temperaturas_medias_ciudad1[2] = 2

var temperaturas_medias_ciudad2 = new Array (3)
temperaturas_medias_ciudad2[0] = 10
temperaturas_medias_ciudad2[1] = 8
temperaturas_medias_ciudad2[2] = 10

```

Con las anteriores líneas se han creado tres arrays de 1 dimensión y tres elementos, como los que ya conocíamos. Ahora crearemos un nuevo array de tres elementos e introduciremos dentro de cada una de sus casillas los arrays creados anteriormente, con lo que tendremos un array de arrays, es decir, un array de 2 dimensiones.

```

var temperaturas_cuidades = new Array (3)
temperaturas_cuidades[0] = temperaturas_medias_ciudad0
temperaturas_cuidades[1] = temperaturas_medias_ciudad1
temperaturas_cuidades[2] = temperaturas_medias_ciudad2

```

Vemos que para introducir el array entero hacemos referencia al mismo sin paréntesis ni corchetes, sino sólo con su nombre. El array `temperaturas_cuidades` es nuestro array bidimensional.

También es interesante ver cómo se realiza un recorrido por un array de dos dimensiones. Para ello tenemos que hacer un recorrido por cada una de las casillas del array bidimensional y dentro de estas hacer un nuevo recorrido para cada una de sus casillas internas. Es decir, un recorrido por un array dentro de otro.

El método para hacer un recorrido dentro de otro es colocar un bucle dentro de otro, lo que se llama un bucle anidado. Puede resultar complicado el hacer un bucle anidado, pero nosotros ya hemos tenido ocasión de practicar en un capítulo anterior. Así que en este ejemplo vamos a meter un bucle FOR dentro de otro. Además, vamos a escribir los resultados en una tabla, lo que complicará un poco el script, pero podremos ver cómo construir una tabla desde javascript a medida que realizamos el recorrido anidado al bucle.

```

document.write("<table      width=200      border=1      cellpadding=1
cellspacing=1>");
for (i=0;i<temperaturas_cuidades.length;i++){
  document.write("<tr>")
  document.write("<td><b>Ciudad " + i + "</b></td>")
  for (j=0;j<temperaturas_cuidades[i].length;j++){
    document.write("<td>" + temperaturas_cuidades[i][j] + "</td>")
  }
}

```

```
        document.write("</tr>")
    }
    document.write("</table>")
```

Este script resulta un poco más complejo que los vistos anteriormente. La primera acción consiste en escribir la cabecera de la tabla, es decir, la etiqueta <TABLE> junto con sus atributos. Con el primer bucle se realiza un recorrido a la primera dimensión del array y se utiliza la variable *i* para llevar la cuenta de la posición actual. Por cada iteración de este bucle se escribe una fila y para empezar la fila se abre la etiqueta <TR>. Además, se escribe en una casilla el número de la ciudad que se está recorriendo en ese momento. Posteriormente se pone otro bucle que va recorriendo cada una de las casillas del array en su segunda dimensión y se escribe la temperatura de la ciudad actual en cada uno de los meses, dentro de su etiqueta <TD>. Una vez que acaba el segundo bucle se han impreso las tres temperaturas y por lo tanto la fila está terminada. El primer bucle continúa repitiéndose hasta que todas las ciudades están impresas y una vez terminado se cierra la tabla.

Inicialización de arrays

Para terminar con el tema de los arrays vamos a ver una manera de inicializar sus valores a la vez que lo declaramos, así se puede realizar de una manera más rápida el proceso de introducir valores en cada una de las posiciones del array.

El método normal de crear un array vimos que era a través del objeto Array, poniendo entre paréntesis el número de casillas del array o no poniendo nada, de modo que el array se crea sin ninguna posición. Para introducir valores a un array se hace igual, pero poniendo entre los paréntesis los valores con los que deseamos rellenar las casillas separados por coma. Veámoslo con un ejemplo que crea un array con los nombres de los días de la semana.

```
var diasSemana = new Array ("Lunes", "Martes", "Miércoles,", "Jueves",  
"Viernes", "Sábado", "Domingo")
```

El array se crea con 7 casillas, de la 0 a la 6 y en cada casilla se escribe el día de la semana correspondiente (Entre comillas porque es un texto).

Ahora vamos a ver algo más complicado, se trata de declarar el array bidimensional que utilizamos antes para las temperaturas de las ciudades en los meses en una sola línea, introduciendo los valores a la vez.

```
var temperaturas_cuidades = new Array(new Array (12,10,11), new  
Array(5,0,2),new Array(10,8,10))
```

En el ejemplo se introduce en cada casilla del array otro array que tiene como valores las temperaturas de una ciudad en cada mes.

Hasta aquí hemos visto la mayor parte de la sintaxis y forma de funcionar de el lenguaje Javascript. Ahora podemos escribir scripts simples que hagan uso de variables, funciones,

arrays, estructuras de control y toda clase de operadores. Con todo esto conocemos el lenguaje, pero aun queda mucho camino por delante para dominar Javascript y saber hacer todos los efectos posibles en páginas web, que en definitiva es lo que interesa

ACTIVIDAD...

Desarrollar una página web en la cual se muestre el funcionamiento de las CSS y de Javascript

ESTRATEGIAS METODOLÓGICAS

- Presentación de la unidad a cargo del profesor.
- Actividades desarrolladas en sala.

RECURSOS

- Humanos: Profesor y alumnos
- Institucionales: Salón de clase, sala de computo
- Materiales: Texto guía

INDICADORES DE EVALUACIÓN

- Evaluación escrita sobre la unidad
- Evaluación de las actividades desarrolladas en sala.

CRITERIOS DE EVALUACIÓN

- ¿Qué son las páginas de estilo en cascada (CSS)?
- ¿Qué funcionalidad tiene el uso de Javascript?
- ¿Cómo podemos utilizar Javascript para mejorar nuestras páginas?